



Infor Cloverleaf Integration Services

Release 2022.09

DRAFT

DRAFT

Important Notices

The material contained in this publication (including any supplementary information) constitutes and contains confidential and proprietary information of Infor.

By gaining access to the attached, you acknowledge and agree that the material (including any modification, translation or adaptation of the material) and all copyright, trade secrets and all other right, title and interest therein, are the sole property of Infor and that you shall not gain right, title or interest in the material (including any modification, translation or adaptation of the material) by virtue of your review thereof other than the non-exclusive right to use the material solely in connection with and the furtherance of your license and use of software made available to your company from Infor pursuant to a separate agreement, the terms of which separate agreement shall govern your use of this material and all supplemental related materials ("Purpose").

In addition, by accessing the enclosed material, you acknowledge and agree that you are required to maintain such material in strict confidence and that your use of such material is limited to the Purpose described above. Although Infor has taken due care to ensure that the material included in this publication is accurate and complete, Infor cannot warrant that the information contained in this publication is complete, does not contain typographical or other errors, or will meet your specific requirements. As such, Infor does not assume and hereby disclaims all liability, consequential or otherwise, for any loss or damage to any person or entity which is caused by or relates to errors or omissions in this publication (including any supplementary information), whether such errors or omissions result from negligence, accident or any other cause.

Without limitation, U.S. export control laws and other applicable export and import laws govern your use of this material and you will neither export or re-export, directly or indirectly, this material nor any related materials or supplemental information in violation of such laws, or use such materials for any purpose prohibited by such laws.

Trademark Acknowledgements

The word and design marks set forth herein are trademarks and/or registered trademarks of Infor and/or related affiliates and subsidiaries. All rights reserved. All other company, product, trade or service names referenced may be registered trademarks or trademarks of their respective owners.

Publication Information

Release: Infor Cloverleaf Integration Services 2022.09

Publication Date: September 28, 2022

Document code: clis_2022.09_clfisoh_op_en-us

Disclaimer

This document reflects the direction Infor may take with regard to the specific product(s) described in this document, all of which is subject to change by Infor in its sole discretion, with or without notice to you. This document is not a commitment to you in any way and you should not rely on this document or any of its content in making any decision. Infor is not committing to develop or deliver any specified enhancement, upgrade, product or functionality, even if such is described in this document.

Contents

Contacting Infor.....	38
Architecture.....	39
File locking.....	39
CIS components.....	39
Core components.....	40
Multi-process and multi-thread design.....	41
Tcl.....	42
Boot-time customization.....	42
Monitoring and control.....	43
Site monitoring.....	43
Rebranding CIS and Security Server.....	45
Revision control.....	46
Configuring the network.....	47
Implementation approaches.....	47
Before implementation.....	48
System configuration.....	49
Command line license help tool.....	50
License help audit report.....	52
License help delimited audit report.....	52
License help information storage.....	53
License help command usage.....	54
Delimited report.....	55
License help examples.....	55
Directory hierarchy.....	56
integrator subdirectories.....	56
\$HCISITEDIR site-specific subdirectories.....	59

\$HCISITEDIR exec subdirectories.....	60
Cloverleaf Integration Services IDE.....	63
Site Manager.....	64
Search functionality.....	64
Open as read-only.....	65
Site Init dialog box.....	65
Creating a site in Windows.....	66
Creating a site in UNIX.....	66
Symbolic links.....	67
Creating a symbolic link to a moved folder.....	67
Creating a symbolic link for smatdb files that are outside HCIROOT.....	68
Loading SMAT history outside the SMAT folder.....	68
Using SMAT/SMAT database when the site is a symbolic link.....	69
Command line usage for sites.....	69
Version control.....	70
Version folders and database.....	71
Site Manager context menu.....	71
Changing the mode.....	73
Comparing versions.....	73
Version status.....	74
Version Control notifications.....	74
Examples.....	75
Client preferences.....	76
Font and document options.....	77
Advanced Network and JVM options.....	77
XSession options.....	78
Xlate Config options.....	80
External commands.....	81
NetConfig/NetMonitor options.....	83
Client SMAT options.....	84
Alert config options.....	85
Adding external tools to the IDE.....	86
Launch bar tools options.....	86
Script editor options.....	86
Right-click menu.....	87

Site preferences.....	88
Tcl log message alignment.....	88
General options.....	88
Log History options.....	89
NetConfig options.....	90
Site SMAT options.....	90
System Management options.....	94
Error handling.....	96
Alert configuration options.....	97
Database Configurations options.....	99
Internal/Error database options.....	103
External database options.....	104
Aliases options.....	105
Statistic database options.....	106
Scheduler.....	110
Configuring Windows authentication through the SQL server.....	112
Connecting to a MySQL database.....	113
Root preferences.....	114
Master site.....	114
Engine output configurations.....	117
Server interface.....	117
Save the references of shared configurations into configuration files.....	121
RootInfo access control.....	121
Data Encoding tab.....	122
File transfer.....	122
Transferring data.....	123
Selecting the environment.....	123
Creating a new directory.....	123
Moving a folder or file.....	123
Renaming a folder or file.....	123
Cloverleaf reconnect.....	124
Emergency save.....	124
Performing an emergency save and restoration.....	124
Reconnecting the client and host.....	125
Client cache.....	125

Cloverleaf ini settings.....	126
server.ini.....	126
security.ini.....	131
client.ini.....	133
Cloverleaf wizard.....	139
Creating and editing a project.....	140
Setting up the wizard.....	141
Auto-Start scripts.....	141
SAML(Ming.le).....	142
Scheduler Log.....	143
Configuring localization.....	144
Infor Ming.le® integration.....	145
Translations.....	146
Adding new variants.....	148
NetConfig metadata.....	148
Examples.....	148
CLWizard/CLAPI site/BOX configuration workflow.....	150
Configuring a CLWizard/CLAPI site/BOX using CLAPI and CLWizard.....	150
Configuring a CLWizard/CLAPI site/BOX using the IDE.....	153
Creating a BOX template.....	154
Cloverleaf API.....	155
RESTful API.....	155
Common issues for CLAPI usage.....	156
Using the API.....	157
Security.....	157
Use cases.....	159
Runtime tools.....	160
Connection Dump.....	161
Viewing the connection dump.....	161
Connection information.....	161
Database Administrator.....	163
Database Administrator definition.....	164
Message and error states.....	168
Database Administrator command-line options.....	171

Disk-based messages.....	172
Configuring disk-based messages.....	173
Disk space.....	173
Using the recovery database.....	174
Using the error database.....	174
Initializing databases.....	174
Error database tool.....	175
Tcl keyed list.....	176
Command line usage.....	176
Error database schemas.....	177
Error trap TPS.....	179
Network Monitor.....	181
Network Monitor Properties dialog box.....	182
Configuring the Engine Output tab.....	184
Setting the protocol status color.....	185
Alerts.....	186
Show thread bitmap.....	188
View list panel.....	188
Grid cell menu.....	189
Process Monitor function.....	190
Managing the view.....	193
Managing threads.....	199
Message resend.....	204
Releasing messages of a process.....	205
Releasing messages of a thread.....	206
Declaring variables.....	206
Finding the next messages to be sent out on a thread.....	207
Performance metrics graphs.....	208
Reviewing performance.....	208
Reviewing the status.....	210
Remote commands tool.....	212
Remote commands GUI options.....	213
Wildcard support in Remote Commands.....	214
Creating and using user-defined commands.....	215
Shell window.....	216

Site Daemons.....	216
Site Daemons tool.....	217
SMAT.....	217
Message states.....	218
Saved message file usage.....	218
Understanding the current message.....	219
Loading multiple files.....	220
View definition.....	221
Using regular expressions for searching SMAT.....	224
Configuring message metadata.....	225
File synchronization.....	227
Resending temp files.....	228
resend command.....	228
resend_errordb and hciddbump commands.....	229
SMAT encoding and temporary files.....	230
Saved message format.....	233
SMAT history.....	236
SMAT secondary.....	239
Saved message file management.....	240
Resending messages.....	240
Message formats.....	242
Monitoring parsing progress.....	243
SMAT database.....	243
Advanced criteria.....	245
SMAT database cycling.....	245
SMAT Database auto-cycling based on age.....	246
crontab SMAT Database cycling.....	247
Recovering corrupted SMAT database data.....	252
Search results.....	252
Deleting SMAT Database messages.....	253
Configuring the display message content size.....	253
Message metadata.....	254
SMAT encryption.....	256
SMAT database encryption key customization.....	258
hcismatcrypt tool usage.....	259

SQLite error log.....	260
Migrating SMAT database files.....	260
Migrating encrypted SMAT databases.....	261
RegEx limitations.....	271
External message flags bit masks.....	272
SMAT database API.....	273
Searching SMAT Database messages with EBCDIC encoding.....	274
Tcl SMAT API.....	274
Configuration tools.....	289
Alert Configurator.....	292
Configuration validation.....	292
Alert types.....	293
Deactivating and activating alerts.....	304
Inserting variables in the tcl, exec, and email actions.....	314
Customizing alert messages.....	319
Alert Time Window dialog box.....	322
Alert file check.....	325
Escalating an alert.....	325
Select Command dialog box.....	328
Starting the Monitor Daemon.....	328
Alert samples.....	331
BOX Manager.....	342
Dependencies between Tcl files.....	342
BOX Manager GUI.....	343
Select Host dialog box.....	344
User-defined folders.....	345
BOX search.....	345
BOX Property dialog box.....	346
Creating a new BOX.....	348
BOX refresh.....	351
How to BOX java driver plug-in threads.....	353
Exporting BOX as a file package.....	354
Importing a BOX package to the current host.....	354
Transferring a BOX between hosts.....	354
Deploying a BOX into the current site.....	355

Deploying a BOX to an existing project.....	357
Merge BOX Resources into Current Site wizard page.....	358
Database protocols.....	358
Resolving conflicts.....	358
Port selection.....	361
Environment-related settings.....	361
Creating a new site with an existing BOX.....	361
BOX folder structure.....	362
Command line usage.....	364
Restoring a site after deploying a BOX.....	364
Database Schema Configurator.....	365
BLOB and CLOB.....	366
Updating a database configuration.....	367
Column and table names.....	368
Importing table/view schema.....	368
Synchronizing the selected table schema.....	368
Synchronizing all table schemas.....	369
Generating database schema from SQL statements.....	369
Adding/Updating SQL statement schema.....	370
Editing table schema.....	370
Table schema options.....	371
Inserting and updating DATE/DATETIME database data.....	371
Examples of using database schema in the Database Inbound protocol.....	372
Examples of using database schema in the Database Outbound protocol.....	373
Using SQL statements.....	374
Engine Output configurator.....	375
EO alias configuration.....	376
Applying aliases with the Network Configurator.....	384
Viewing engine output.....	384
FRL Configurator.....	385
Configuring the field definition.....	386
Configuring field subfield properties.....	387
Configuring a mask.....	388
Configuring mask subfield properties.....	389
FRL group definition.....	390

Configuring group field properties.....	390
Configuring group subfield properties.....	391
FRL data types.....	391
FRL fill characters.....	392
FRL record layout definition example.....	395
Global Variables Configurator.....	396
Data flow.....	397
Information storage.....	398
Encryption.....	398
BOX Manager global variables.....	398
Validation in other tools.....	399
Tcl and Java interfaces for global variables.....	399
Global variables Tcl interface usage.....	399
Global variable example.....	401
HL7 Configurator.....	402
CCD support.....	402
HL7 standard.....	403
HL7 fields.....	407
HL7 segments.....	407
HL7 messages.....	408
HL7 format definition example.....	410
HPRIM Configurator.....	411
Segment combining and splitting.....	412
HPRIM fields.....	413
HPRIM data types.....	414
HPRIM segments.....	415
HPRIM messages.....	416
HPRIM definition example.....	417
HRL definition.....	418
Condition options.....	419
Repeat options.....	419
Using Repeat While example.....	421
Using Repeat Block example.....	421
Empty segment handling.....	422
Import Script.....	423

JSON Configurator.....	424
FHIR support.....	424
Reference nodes.....	425
JSON translation support.....	425
Configuration modes "Type" and "Ref".....	426
Combining schemas.....	427
Using list and tuple for array.....	427
Required, Validation, and Ordering options.....	428
Configuring JSON in Cloverleaf.....	430
Creating a JSON definition by manually editing nodes.....	430
Creating a JSON definition based on a JSON message.....	430
Testing the JSON message against a definition.....	431
Configuring a translation for the JSON format.....	431
JSON use cases.....	432
Determining the Trx ID on a JSON message.....	433
JSON Schema Configurator.....	433
JSON Schema Configurator user interface.....	434
JSON schema definitions.....	435
JSON Pointer.....	436
JSON schema reference.....	437
Loading JSON schema.....	438
Converting JSON data to JSON schema.....	438
LDL Configurator.....	439
LDL definition files.....	440
Configuring an LDL field.....	440
Configuring an LDL segment.....	441
Configuring an LDL message.....	442
Limitations in message variations.....	443
IOD (Information Object Definition).....	444
DICOM and HL7.....	445
Additional information in message metadata.....	446
Logging and debugging.....	446
Handling large messages.....	449
DICOM definition generate tool.....	449
Lookup Table Configurator.....	450

Lookup table definition: Table lookup.....	450
Lookup table definition: Database lookup.....	451
Error handling of database lookup.....	455
Multiple rows returned from the database lookup TABLE action.....	456
Database lookup Tcl extension.....	457
Using multi-byte encoding in the Lookup Table.....	457
HIPAA external code sets.....	457
Sorting and searching table items.....	458
Table security.....	458
Reserved table names.....	461
Lookup Table configuration example.....	463
NCPDP Formulary and Benefit Configurator.....	464
Copying and pasting across variants.....	464
Batch types.....	464
Message definition files.....	470
NCPDP Formulary and Benefit standard.....	471
Data elements.....	471
Data segments.....	472
Messages.....	473
NCPDP Formulary and Benefit format definition example.....	474
NCPDP Telecom Configurator.....	475
NCPDP Telecom message structure.....	476
NCPDP Telecom batch standard.....	481
Fields.....	483
Composite data structures.....	484
Tokens.....	484
Transactions.....	485
NCPDP Telecom format definition example.....	486
Network Configurator.....	488
User interface.....	489
Reading/editing master site objects from another site.....	490
Enabling/Disabling procs in the TPS editor.....	490
Thread icons.....	490
TPS Editor language support.....	493
Process configuration dialog box.....	493

Thread template.....	499
Multi-threaded translation.....	501
Message tracing.....	504
Message tracing examples.....	509
Clearing the tracing database.....	514
SRCMID.....	514
SOURCECONN and ORIGSOURCECONN.....	514
NetConfig.....	515
siteInfo.....	515
Leveraging CIS in a service-oriented architecture.....	516
Outbound tab.....	520
Properties tab.....	543
Inbound tab.....	550
Route Replies tab.....	557
Route Messages tab.....	558
Notes tab.....	558
Verifications tab.....	559
Transaction Summary tab.....	559
Creating routes.....	560
Route configuration.....	561
Reroutes.....	562
Editing operations.....	565
Protocol properties configuration.....	566
Thread inbound data configuration.....	566
Route detail property configuration.....	566
Outbound thread properties configuration.....	566
Moving threads and routes.....	567
Route representation.....	567
Viewing routes and threads.....	567
HCI_static_else_route.....	571
Inter-site routing.....	571
Wild card routing example.....	575
Route details.....	576
Configuring a reply route.....	582
Configuring, customizing, and migrating message archiving.....	585

Examples of using Network Configurator.....	587
Protocols.....	589
IPv6.....	589
Database protocols.....	593
DICOM protocol.....	622
DTC.....	637
Fileset FTP protocol.....	654
File protocol.....	674
Fileset/FTP Local protocol.....	677
HTTP Client protocol.....	686
Generic Java driver.....	693
Server port configuration.....	694
LU 3 protocol.....	695
LU 6.2 APPC protocol.....	695
WebSphere MQ protocol.....	702
PDL drivers.....	715
Delay connection until needed.....	720
TCP/IP protocol.....	720
TCP/IP MLP.....	727
UPoC protocol.....	729
Prosper Async protocol (UNIX).....	732
Link Async protocol (UNIX).....	733
Advanced scheduling.....	734
Scheduling dialog box.....	735
Configuring a new schedule sequence.....	737
Modifying an existing schedule.....	738
Network Monitor timer thread state.....	738
Script Editor.....	738
Script Editor and Tcl procedures.....	739
Selecting a Tcl procedure.....	740
Advanced features.....	740
Client Preferences Script Editor options.....	741
Proc types.....	742
Script Editor search options.....	744
Tcl namespace support.....	744

User-defined Tcl templates.....	745
External Editor and Import Script tool.....	745
Tcl editor use cases.....	746
Site Document.....	747
Site Document GUI.....	747
Server interface.....	748
Changing the Site Document URL to a different path.....	748
Generating and viewing the Site Document.....	749
Translation Configurator.....	750
Translation pipeline.....	751
Translation configuration.....	751
Address indicator node.....	754
XML and the Translation Configurator.....	755
Message flow between threads.....	771
Source and destination values.....	771
"%" variable names.....	774
Configuring a database schema format.....	774
XML support and characteristics.....	775
GUI behavior.....	775
Error handling.....	777
Translation scripting.....	777
Xlate Debugger.....	781
Actions.....	790
Action properties.....	816
Translation Configurator examples.....	818
UN/EDIFACT Configurator.....	831
Encoding separators.....	832
UN/EDIFACT message formats.....	832
UN/EDIFACT data elements.....	834
UN/EDIFACT composite data structures.....	834
UN/EDIFACT data segments.....	835
UN/EDIFACT messages.....	836
Configuring a UN/EDIFACT variant definition.....	837
VRL Configurator.....	838
VRL definition format.....	840

Configuring the VRL field.....	841
Configuring the VRL subfield properties.....	842
Configuring the VRL subsubfield properties.....	843
Configuring the VRL mask.....	844
Configuring the VRL mask subfield properties.....	845
Configuring the VRL mask subsubfield properties.....	845
Configuring the VRL group field properties.....	846
Configuring the VRL group subfield properties.....	847
Configuring the VRL group subsubfield properties.....	848
VRL record layout definition example.....	848
WSDL2XSD tool.....	850
X12 Configurator.....	850
Syntax notation.....	851
HIPAA.....	854
Configuring an X12 data element.....	855
Configuring the X12 composite data structure.....	855
Configuring the X12 data segment.....	856
Configuring the X12 transaction set.....	856
Creating optional and repeating groups.....	857
XML Package Manager.....	858
XML terms.....	859
User operations.....	860
Root names.....	860
XML Package Manager security.....	861
XML definition files.....	862
File management.....	862
Compiling/Recompiling.....	866
hcxmlcompile usage.....	867
HL7 v3 CDA 2.0 support.....	872
Compile Options dialog box.....	874
XML Compile Properties dialog box.....	875
Schema support.....	875
Testing Tool output.....	876
Namespace support.....	876
Moving an XSD file to host and compiling.....	879

Index files.....	880
Testing tools.....	881
BLOB/CLOB support in Testing Tool.....	883
BLOB/CLOB command line support.....	883
Lookup table extension.....	883
System topology.....	884
System tests.....	884
Unit testing.....	884
Integration and system testing.....	885
Test output.....	885
Testing the database protocol.....	886
Testing the database schema.....	887
Testing DICOM configurations.....	888
Testing EDIFACT configurations.....	889
Testing FRL configurations.....	890
Testing HL7 configurations.....	890
Testing HPRIM configurations.....	891
Testing HRL configurations.....	892
Testing JSON configurations.....	893
Testing LDL configurations.....	894
Testing routes.....	895
Testing TPS.....	897
Testing a transaction ID.....	899
Testing VRL configurations.....	900
Testing X12 configurations.....	900
Testing NCPDP Telecom configurations.....	902
Testing NCPDP SCRIPT configurations.....	903
Testing NCPDP F&B configurations.....	904
Testing XLT configurations.....	905
Testing XML configurations.....	906
Testing XSLT configurations.....	907
Using XSLT files to validate.....	909
XSLT examples.....	909
TCP/IP.....	911
Verifying external interfaces.....	911

Testing TCP/IP connections.....	912
HIPAA external code sets.....	914
HIPAA guide number 5: Countries, currencies, and funds.....	914
HIPAA guide number 22: States and outlying areas of the U.S.....	914
HIPAA guide number 130: Health care financing administration common procedural coding system.....	915
HIPAA guide number 131: International classification of diseases clinical mod (ICD-9-CM) procedure.....	916
HIPAA guide number 132: National uniform billing committee (NUBC) codes.....	916
HIPAA guide number 133: Current procedural terminology (CPT) code.....	916
HIPAA guide number 134: National drug code.....	917
HIPAA guide number 135: American dental association code.....	919
HIPAA guide number 139: Claim adjustment reason code.....	920
HIPAA guide number 235: Claim frequency type code.....	920
HIPAA guide number 240: National drug code by format.....	920
HIPAA guide number 245: National association of insurance commissioners (NAIC) code.....	920
HIPAA guide number 507: Health care claim status category code.....	925
HIPAA guide number 513: Home infusion EDI coalition (HIEC) product/service code list.....	925
HIPAA guide number 540: Health care financing administration national plan ID (subject to availability).....	926
HIPAA guide number (no number): Provider taxonomy code.....	926
PDL.....	927
States.....	928
Write operations.....	928
Read operations.....	929
Components of a PDL driver.....	929
Structure of a PDL driver.....	930
Declarative section.....	930
Extended Backus-Naur Form grammar.....	931
Tcl code section.....	932
Encodings.....	934
Phrase-constant parts.....	935
Fixed-array parts.....	935
Variable-array parts.....	936
Fields.....	936
Phrase-checked parts.....	937

Length-encoded phrase parts.....	938
Available Tcl functions.....	939
hci_pd_receive contns.....	939
hci_pd_send phrase data contns.....	941
hci_pd_accept info.....	942
hci_pd_deliver.....	943
hci_pd_open_device [contns[config]].....	945
hci_pd_report_exception code text.....	946
hci_pd_set_result_code code.....	946
uassert expr [descr].....	947
Built-in PDL functions.....	947
Named character classes.....	949
Sample PDL device definition for a TCP driver.....	949
Sample PDL device definition for an async driver.....	950
Compiling PDL code.....	952
Cloverleaf multi-byte encoding.....	955
Limitations of unicode.....	956
Remote machines in a different operating system.....	957
Handling of unsupported encodings.....	957
Handling of invalid byte streams.....	957
msgfinduchar: Maximum permissible character value.....	958
Non-ASCII data converted to UTF-8.....	958
Command-line utilities.....	959
hxicmd usage.....	959
Restrictions.....	959
hcitcl.....	960
hciwish.....	960
cvtnull option.....	960
Test steps.....	960
Null characters in messages.....	961
Relationship between encoding conversion and UPoCs.....	961
Relationship between encoding conversion, the recovery database, SMAT, and resend.....	961
Backward compatibility of SMAT files.....	962
Fixed encoding.....	963
Fixed encoding with "Use byte order mark".....	963

XML encoding.....	963
XML encoding with "Generate byte order mark".....	964
XML processing behavior change.....	964
Example: XML Cyrillic ISO8859-5 to UTF-8 translation.....	964
Example: XML Cyrillic ISO8859-5 to UTF-8 translation in the system Unicode.....	965
Configuration tips.....	966
Configuration of language options in Windows.....	967
Configuring the regional language setting in English operating systems.....	967
Configuring the Windows multi-language edition.....	967
Configuration of language options in Linux.....	968
Redhat Linux installation with only English installed.....	968
Encoding conversion between the IDE and the engine.....	969
Tcl UPoC.....	969
Java UPoC.....	970
UPoC protocol.....	970
UPoCs.....	971
Scripting languages.....	972
TPS module interface.....	973
Keyed lists.....	973
Contexts.....	974
Message dispositions.....	975
UPoCs used within the TPS module interface.....	976
Code fragment interface.....	984
Code fragment UPoCs.....	984
Transaction ID determination interface.....	985
Guidelines.....	986
Using HAPI.....	987
UPoC Debugger solution.....	987
CLDI.....	991
Magic token (authentication request).....	992
Debugging Tcl UPoC with NetBeans.....	992
Debugging Java/JavaScript UPoC with NetBeans.....	996
Debugging Python UPoC.....	997
Debugging notifications.....	997
Debugging commands.....	998

The UPoC Java package.....	1001
Java equivalents.....	1002
Script UPoC Java Embedded Python.....	1003
Architecture with UPoC Java.....	1004
Constructors and userArgs.....	1004
Instances and construction.....	1005
Destroy method.....	1005
Use of class (static) variables.....	1005
Example.....	1005
Generic Java Driver V2.....	1006
GJD V2 design.....	1007
Multiple javadriver threads.....	1008
thread_name.ini and process_name.ini.....	1009
Class path "path/*.jar".....	1009
Generic Java driver V2 FAQ.....	1009
Directories and sample files.....	1013
Additional directories.....	1013
Environment setup.....	1014
Java: Transaction ID determination interface.....	1014
Examples.....	1014
Java: Code fragment interface.....	1015
Translation actions and UPoC SubClasses.....	1015
Examples.....	1016
Java: TPS module interface.....	1019
Examples.....	1020
Helper classes.....	1024
Message class.....	1024
DispositionList.....	1025
GRM and its subclasses.....	1028
XPM.....	1029
PropertyTree.....	1029
MsgDataInputStream and MsgDataReader.....	1030
MsgDataOutputStream and MsgDataWriter.....	1030
Exceptions.....	1030
Exception notification mechanism.....	1031

Configuration.....	1031
Usage.....	1032
Utility and convenience methods.....	1032
Using Java code from Tcl UPoCs.....	1033
Calling a Java TPS from Tcl.....	1033
Calling a Java TrxID from Tcl.....	1034
Calling a Java XLT from Tcl.....	1035
JDDK.....	1036
Java driver debug example.....	1037
Configuration.....	1039
Thread template.....	1044
JVM auto defines.....	1044
.ini and .pni entries.....	1044
Usage.....	1053
Usage patterns.....	1053
Javadoc for Java Driver API.....	1054
Sample applications.....	1054
WeatherClient.....	1055
Bouncefile and Bounce (pattern 1).....	1055
RequestServer (pattern 3.c).....	1057
MessageServer (pattern 2.a).....	1059
MessageServer (pattern 2.b).....	1060
RequestServer (pattern 3.a).....	1061
RequestServer (3.b Unique Callback).....	1061
CallbackJava/Bounce (pattern 3.b Registered Callback).....	1063
CallMeBack.java.....	1064
JNI interface.....	1066
Minimizing inbound and outbound JVMs.....	1067
Development procedure.....	1068
Logging in Java driver.....	1068
Script UPoC.....	1069
LANG, FILE, FUNC, and SCRIPT keys.....	1069
Jar files and Script UPoC classes.....	1070
JavaScript UPoC.....	1071
JavaScript scripting in UPoC.....	1072

Python UPoC.....	1072
Calling functions defined in another file.....	1073
Working with GitHub in Cloverleaf application development.....	1080
Developing Cloverleaf with GitHub.....	1080
Developing the Cloverleaf site with GitHub.....	1082
GitHub reference: gitattributes.....	1083
GitHub reference: Site-level gitignore.....	1084
GitHub reference: Root-level gitignore.....	1085
Server administration.....	1087
Setting the environment.....	1088
Working with sites.....	1088
Command-line users.....	1088
Renaming a site.....	1088
Security.....	1089
Web-based server administration.....	1089
Advanced security host server settings.....	1089
Updating certificate passwords.....	1090
Host server administration.....	1090
Run menu password options.....	1090
Host server certificate password update.....	1090
Security server certificate password update.....	1091
Host server and security server certificates password update.....	1091
Email setting dialog box.....	1091
Host Server General tab.....	1092
Host Server Firewall tab.....	1092
TCP/IP settings.....	1092
Net address translation port.....	1093
When GUI access is not available.....	1093
Host Server Advanced tab.....	1094
Open Tools options.....	1095
XLT debug traffic.....	1096
JVM options.....	1096
Log-out options.....	1097
Heap size.....	1097

Security Server RMI registry port.....	1097
Host Server Monitor tab.....	1098
Suspended User Cache tab.....	1098
Host Server Web Server tab.....	1099
Web services and security.....	1100
Common web services.....	1100
SMAT message resend related web services.....	1101
Error message resend related web services.....	1107
Local binding support for CAA-WS.....	1111
Using the route command.....	1112
Host Server Encryption tab.....	1118
Host Server LDAP tab.....	1119
Configuring LDAP authentication.....	1119
LDAP Advanced Configuration dialog box.....	1121
Groups.....	1122
Account exception list.....	1123
Create user certificate file and private key file automatically.....	1123
Host Server Launch Bar Tools tab.....	1123
User exception List.....	1123
Host Server Version Control tab.....	1124
Host Server Databases tab.....	1124
Host Server Certificate tab.....	1126
Allowlist tab.....	1126
Predefined and user-defined allowlists.....	1127
Command validation.....	1128
Java validation.....	1129
Using CLAPI in allowlist management.....	1129
Allowlist examples.....	1131
Security Audit tool.....	1134
Security Audit options.....	1135
Audit report.....	1136
Audited items.....	1136
hcisecurityaudit command usage.....	1144
hcverify command usage.....	1144
Report examples.....	1145

Security Audit example.....	1147
System ports.....	1147
Port range.....	1148
Security Server Firewall tab.....	1148
Generating SSL certificates.....	1149
Security Server Advanced tab.....	1149
Security Server LDAP tab.....	1150
Synchronization section.....	1150
LDAP Advanced Configuration dialog box.....	1151
LDAP information storage.....	1152
Troubleshooting the LDAP configuration.....	1152
Security Server Encryption tab.....	1153
Configure Encryption Keys panel.....	1154
Use cases.....	1155
Security Server Web Server tab.....	1155
Security Server Database tab.....	1156
System maintenance.....	1158
Auto-Start.....	1158
Auto-Start environment.....	1159
Server interface.....	1159
Auto-Start Scripts options.....	1160
Cloverleaf auto-healing toolset.....	1161
Cloverleaf Cluster.....	1163
Failover support.....	1166
Auto-Start deployment.....	1167
Auto-Start use cases.....	1168
System backup.....	1168
Cycling files.....	1169
File system maintenance.....	1169
Database directory maintenance.....	1169
Database file location.....	1170
UNIX system solutions.....	1170
Windows system solutions.....	1171
Management tips.....	1171
Automating maintenance.....	1171

Automating maintenance in UNIX.....	1171
Automating maintenance in Windows.....	1172
Tcl leaks.....	1173
Setting HTTP and SOAP headers in CAA-WS Clients.....	1174
SOAP extension configurations.....	1174
Adding arbitrary HTTP headers in messages by overriding the userdata items.....	1175
Adding arbitrary SOAP headers for SOAP Consumers.....	1176
Configuration tips.....	1176
Setting up Security Server ACLs to enable GM functions.....	1178
Cloverleaf security.....	1180
SSL functions.....	1180
Setting up TCP/IP SSL authentication.....	1180
SSL types and modes.....	1181
SSL client modes.....	1182
SSL server modes.....	1183
Mode matches.....	1185
OpenSSL ciphers.....	1185
Cloverleaf default password strength rules.....	1190
Definitions.....	1190
Self-signed certificates.....	1191
Generating an RSA key self-signed root CA.....	1192
Generating an RSA key customer certificate.....	1193
Installing the Host Server and Security Server.....	1193
Upgrading to basic and advanced security.....	1194
Launching the IDE.....	1194
Generating certificates.....	1194
Algorithms.....	1194
Authentication.....	1194
Command line interaction with CIS security.....	1195
siteSecurityInfo file.....	1195
Database encryption: \$HCISITEDIR/siteSecurityInfo.....	1197
Database encryption: hcicreatesite.....	1198
Database encryption: hcisiteinit.....	1199
Database encryption: hcirootcopy.....	1199

Database encryption: hcidbencrypt.....	1200
Encryption.....	1201
Encryption methods.....	1202
Protocol GUI security settings.....	1202
SSL protocol.....	1203
CA Path and CA File.....	1203
Examples of client/server negotiation.....	1204
Upgrading and downgrading security.....	1204
Security levels.....	1205
Security passwords.....	1206
Before upgrading security.....	1206
Upgrading CIS 6.2.x host server to advanced security using CIS 19.1+ security server.....	1207
Upgrading from none to basic security.....	1207
Upgrading from none to advanced security.....	1208
Upgrading from basic to advanced security.....	1209
After upgrading security.....	1210
Changing the security server.....	1211
Removing the security server.....	1211
Downgrading security.....	1211
Security server administration.....	1212
Advanced security.....	1212
Building a system with advanced security.....	1215
Configuring ACLs in Advanced Security mode.....	1215
Security modes.....	1227
Certificates.....	1228
Starting with advanced security.....	1229
Running with advanced security.....	1229
Stopping with advanced security.....	1229
Verifying a server is running.....	1230
Advanced security logging.....	1232
Error messages in the log files.....	1232
Changing modes of security.....	1234
Removing a site in advanced security.....	1235
Nodes and ACLs.....	1235
Use cases.....	1240

ACL/Role Manager.....	1240
Advanced security users and roles.....	1247
Advanced security permissions.....	1250
Multiview.....	1256
Security Server Migration tool.....	1256
Security log in.....	1257
User credentials.....	1257
Preferred licensing method.....	1257
Alternate licensing method.....	1258
PEM format.....	1258
Roles.....	1258
Obtaining a user certificate.....	1258
Certificate Manager.....	1260
Service Setting dialog box.....	1260
Certification.....	1261
Securing the user certification process.....	1262
Certificate states.....	1262
Passwords and the user.....	1263
Audit Log.....	1264
Creating the web services certificate.....	1264
Certificate expiration.....	1264
Updating the CA certificate.....	1265
Updating or renewing issued certificates.....	1265
Starting the Certificate Manager.....	1265
Accessing and logging on to the Certificate Manager.....	1265
Adding new users.....	1266
Logging on.....	1266
Exiting the Certificate Manager.....	1267
Issuing user certificates.....	1267
Issuing user certificates with user participation.....	1268
Issuing user certificates without user participation.....	1269
Requesting a user certificate.....	1271
Issuing JKS certificates.....	1272
Keystore tab.....	1273
Issuing host server certificates.....	1273

Issuing security server certificates.....	1274
Viewing certificate information.....	1275
Changing a user password.....	1275
Revoking and un-revoking certificates.....	1276
Revoking user certificates.....	1277
Un-revoking a user certificate.....	1277
Refreshing a CRL or an expired CRL.....	1278
Refreshing the CRL.....	1278
Refreshing an expired CRL.....	1278
Audit Log Viewer.....	1280
Enabling the Audit Log.....	1281
Accessing the Audit Log Viewer.....	1281
Logging on to Audit Log Viewer.....	1281
Exiting the Audit Log Viewer.....	1282
Audit Log Viewer dialog box.....	1282
Sorting log entries.....	1283
Audit log search.....	1283
Audit log export.....	1284
Audit Log detail panel.....	1284
Audit Log preferences.....	1285
Log Size Control pane.....	1287
Log history pane.....	1288
Tracking user activities.....	1288
Audit log cycle files.....	1289
server.ini file.....	1289
clide.ini file.....	1290
Audit log file.....	1291
Logfile of engine events.....	1293
Log history.....	1293
Configuring log history features.....	1294
Configuring JMX on Apache Tomcat.....	1296
CPU affinity in Cloverleaf runtime (Windows/Linux).....	1297
Reference guide.....	1299
Engine components.....	1299

Threads.....	1300
Message flow between threads.....	1300
Queues.....	1301
Tcl interpreter.....	1302
Upgrading from Tcl 7.6.....	1303
Route detail types.....	1306
Raw route detail type.....	1307
Translate detail type.....	1308
Generate detail type.....	1309
Return value.....	1309
State information.....	1310
Message flow.....	1310
Inbound protocol message flow.....	1310
Translation thread message flow.....	1311
Outbound protocol message flow.....	1311
Reply handling.....	1312
Protocol message forwarding.....	1312
Chaining.....	1312
Disk-based messages.....	1313
Recovery database message flags.....	1313
Disk-based queuing and VM/DISKQ_PERCENT environment variable.....	1314
Interprocess message movement.....	1314
Engine commands.....	1315
hci GUI commands.....	1315
hci non-GUI commands.....	1317
Security startup commands.....	1410
Server startup commands.....	1410
Security GUI startup commands.....	1410
Security utility startup commands.....	1410
hci commands.....	1411
Control command.....	1411
Counter commands.....	1412
Datum commands.....	1413
Files commands.....	1415
GDBM commands.....	1415

Global variables commands.....	1420
GRM commands.....	1420
Lists commands.....	1427
Message commands.....	1427
MSI commands.....	1435
Strings commands.....	1437
Table commands.....	1437
Xlate commands.....	1439
XPM commands.....	1442
HTTP commands.....	1446
ibmime commands.....	1454
ibmime Tcl API.....	1455
ibmime usage examples.....	1458
Control headers between Cloverleaf and Intelligent Broker.....	1463
ibmime headers overriding default behavior.....	1466
Sample Tcl procedures.....	1466
Tcl extensions.....	1467
Counter commands and references.....	1468
Datum extensions and references.....	1470
GRM extensions and references.....	1474
International character data support.....	1486
Arbitrary precision math operations.....	1492
Message extensions.....	1494
Message references.....	1508
MSI extensions and references.....	1515
XPM extensions and references.....	1517
Generate routes.....	1524
Protocol thread status.....	1526
Tcl extension modifications.....	1526
Engine NetConfig interface extensions.....	1527
Loading a specified NetConfig.....	1528
Querying information about threads.....	1528
Querying information about processes, groups, or destination threads.....	1529
Retrieving the NetConfig file version.....	1529
Detecting a NetConfig modification.....	1529

NCI reference.....	1530
HTTP server.....	1531
HTTP server interface.....	1531
HTTP server usage.....	1533
grmfetch.....	1536
X12 procedures.....	1536
joinX12.....	1537
splitX12.....	1538
HL7 message types (functional areas).....	1541
HL7 segment ID.....	1544
HL7 type codes.....	1549
Metadata fields.....	1551
System-level environment variables.....	1553
msgXSLT.....	1553
Data Integrator.....	1554
ODBC Tcl extensions.....	1554
Example using catch.....	1555
ODBC Tcl API and C API differences.....	1555
Memory allocation.....	1556
Data types.....	1556
Error returns.....	1556
Coding examples.....	1557
Application example.....	1566
ODBC command reference.....	1570
ODBC commands.....	1570
ODBC deprecated commands.....	1579
Troubleshooting.....	1583
Error database states.....	1583
State 107 note.....	1585
State 416 note.....	1586
Recovery database states.....	1586
Recovery database troubleshooting.....	1587
DB Vista -921 recovery.....	1587
DB Vista -921 user ID check failure.....	1587

DB Vista 940 recovery.....	1588
Resetting the database and status.....	1588
Resetting the database contents.....	1588
Resetting the status.....	1589
Cloverleaf tips.....	1590
HL7 scripts.....	1590
msgExtract.....	1591
msgExtract examples.....	1591
msgReader.....	1592
msgReader examples.....	1593
msgParser examples.....	1597
Using xlateInVals and xlateOutVals.....	1600
Passing multiple values in and from Tcl fragments.....	1601
Accessing Tcl help.....	1602
Fetching value.....	1602
Fetching examples.....	1603
Updating an older translation file to work under the new fetch behavior.....	1604
Using Tcl string map.....	1605
External message flags bit masks.....	1606
Upgrading encodings to MB or 6.2 and later versions.....	1607
Configuring a standard inbound TCP/IP server thread.....	1607
Configuring a standard outbound TCP/IP client thread.....	1608
Using the Fileset protocols without a custom TPS procedure.....	1608
Deleting a single message.....	1609
Using UltraLong format.....	1609
Performing search functions.....	1609
Recovery database lock.....	1610
Setting up UNIX users.....	1610
Adding/removing hciss auto-starting on UNIX/Linux.....	1611
Installing the host server as a service in UNIX/Linux.....	1611
Recovery database.....	1611
Operator guide.....	1613
Threads and processes.....	1613
Components.....	1614

Tools.....	1614
Cloverleaf system.....	1615
Network Configurator.....	1615
Selecting the environment.....	1615
Site Daemons.....	1616
Accessing the site daemons.....	1616
Starting the Site Daemons tool.....	1617
Network Monitor controls.....	1617
Stopping the site daemons.....	1617
Parameter pane.....	1617
Command pane.....	1618
Site daemon operations.....	1618
Site daemons from the command line.....	1620
Network Monitor.....	1621
Starting the host server.....	1622
Starting and stopping the Lock Manager and Monitor Daemon.....	1622
Delay interval.....	1623
MonitorD error status and log.....	1623
Starting Network Monitor.....	1623
Network Monitor tabs.....	1624
View and Options menus.....	1624
Command and engine output.....	1625
Process monitor.....	1626
Alerts.....	1627
Network Monitor Properties dialog box.....	1630
Selecting thread control in a process.....	1634
Process Controls dialog box options.....	1634
Managing views and threads.....	1636
Thread management options.....	1642
Accessing the Thread Control context menu.....	1643
Accessing the Thread Controls dialog box.....	1644
Tracking and reviewing performance.....	1646
Reviewing thread metrics.....	1647
Engine Monitor menu options: Graphs menu.....	1647
Engine Monitor menu options: Interval menu.....	1648

Totals pane.....	1648
Performance metrics graphics.....	1649
Status review.....	1649
Resolving issues.....	1651
Contacting Support.....	1651
Running low on disk space.....	1652
File permission issues.....	1652
Running Cloverleaf with UAC enabled on Windows.....	1653
Processes run without status information.....	1653
Restarting the engine after a power outage.....	1653
Testing TCP/IP connections.....	1654
Resolving a binding error.....	1654
Client does not connect to the server.....	1654
INEOF states.....	1655
db_vista -921 errors.....	1655
CPU is maxed out at 90% or higher.....	1655
Virtual memory requirements.....	1655
"/dev/kmem not world readable" error when launching Network Monitor.....	1655
Java errors.....	1656
Viewing the process directory log files.....	1656
Viewing the recovery and the error databases.....	1656
Command-line utilities.....	1657
hcicmd.....	1657
hconndump.....	1657
hconnstatus.....	1658
hcidbdump.....	1658
hciengineerun.....	1660
hcienginerestart.....	1660
hcienginestop.....	1661
hcilmclear.....	1662
hcimsiutil.....	1662
hciprocstatus.....	1663
hcsitecleanup.....	1663
hcsitctl.....	1663
hctcptest.....	1665

hcverify.....	1665
Recovery database troubleshooting.....	1666
DB Vista -921 recovery.....	1666
DB Vista -921 user ID check failure.....	1666
DB Vista 940 recovery.....	1667
Resetting the database and status.....	1667
Resetting the database contents.....	1667
Resetting the status.....	1668
Index.....	1669

Contacting Infor

If you have questions about Infor products, go to Infor Concierge at <https://concierge.infor.com/> and create a support incident.

The latest documentation is available from docs.infor.com or from the Infor Support Portal. To access documentation on the Infor Support Portal, select **Search > Browse Documentation**. We recommend that you check this portal periodically for updated documentation.

If you have comments about Infor documentation, contact documentation@infor.com.

Architecture

The Cloverleaf Integration Services (CIS) engine script can run on a variety of Windows and UNIX platforms. This engine receives messages from external systems through protocols such as TCP/IP, FTP, and Fileset. It then performs transformations on the messages, and sends the messages to their destination through any of the same set of supported protocols.

The Cloverleaf engine records statistics to shared memory, where it can be accessed by a separate monitor daemon process. This daemon makes these statistics available to the host server through TCP/IP and ultimately available to users running the Cloverleaf IDE through the host server. The daemon also provides an alert mechanism, which can alert the user when user-defined trigger conditions occur.

The Cloverleaf host server acts as a bridge between any number of Cloverleaf IDEs and the engine and monitor Daemon. The host server interacts with the Cloverleaf IDE through RMI and accesses the Cloverleaf engine through the local file system and monitor daemon through TCP/IP.

Advanced security can be configured through the Cloverleaf security server. Residing on a separate machine from any host servers accessing it, the security server keeps access control lists on an embedded Derby database. Whenever a user attempts to perform a task, the host server queries the security server through RMI. It does this to determine whether the user has permission to perform the task.

File locking

Advisory locking is used on configuration files. Locking prevents multiple users from editing the same file simultaneously. File locks are enabled when a configuration file is opened for editing within the IDE. No file locks are enabled from the command line.

A single lock file is created in the `HCISITEDIR` lock directory for each configuration file that is opened. This file contains information about the user, the process, and the time the file was locked.

If a user attempts to edit a locked file, then a dialog box opens with two options. You can override the lock or load the file in read-only mode.

CIS components

The IDE is a GUI from which users control Cloverleaf on both the local and remote hosts.

The security server prevents unauthorized access with defined user IDs, passwords and access control lists, facilitating HIPAA compliance.

See [Cloverleaf security](#).

The Data Integrator integrates Cloverleaf with databases such as Oracle, Microsoft SQL Server, and IBM DB2 through ODBC.

See [ODBC Tcl extensions](#).

You can use UPoCs to customize Cloverleaf at various access points, by writing scripts in TCL or Java. Scripts can be used to control both message content and flow. TCL scripts for common tasks such as HL7 ack/nak procedures are shipped with the product and can be modified as required. There is also an active community where customers share their scripts with other customers who have similar requirements.

See [UPoCs](#).

Testing tools are GUI and command-line tools for testing message translation configurations and data files without sending messages through the Cloverleaf engine.

See [Testing tools](#) on page 881.

The Cloverleaf generic Java driver provides an API that you can use to create Java applications that directly access Cloverleaf. Through this API, Java is treated as a Cloverleaf protocol, and external Java applications can exchange messages directly with Cloverleaf Java protocol threads.

See [Generic Java Driver V2](#).

All message content passing into and from Cloverleaf can be archived to an external relational database. This is where the data can be accessed for any purpose, such as calculating statistical information or keeping an audit log.

See [Configuring, customizing, and migrating message archiving](#).

Core components

The core components of Cloverleaf consist of these concepts:

- All messages passing through Cloverleaf are stored in queues between any actions that are undergone in the engine. These queues are backed by an embedded high-performance Raima network-model recovery database. This ensures message continuity and integrity can be recovered in the event of an unexpected shutdown by repopulating the queues from the database.

See [Introduction to Database Administrator](#).

Messages that encounter errors are also stored in a similarly designed error database, from where they can be modified and resent to various points in the engine. A third embedded database acts as a cache for message archiving where messages are stored until they are flushed out to an external relational database.

- Routing is the way Cloverleaf determines what messages go where. This is accomplished by creating routes from inbound protocol threads to outbound protocol threads. A Transaction ID that is based on the message's content is defined by the user. This is used to determine which routes a message takes.

Translations can be configured to occur during these routes. An inter-site routing tool is also provided. With this tool, Cloverleaf sites on several hosts can route messages to one another directly using the Interthread Communications Library.

See [Route configuration](#).

- Cloverleaf offers many built-in tools to transform messages through mathematical, logical and iterative operations or lookup tables. The user can define how these tools are used in the IDE's Translation Configurator and save a particular translation into a translation file. These files can be tested in the testing tool to determine whether they work as designed before deploying in a production environment. When these tools alone are insufficient, the user can write scripts in Tcl or Java to transform messages.

See [Introduction to Translation Configurator](#).

- Scripts that control message flow are run using embedded Tcl interpreters and JVMs. Scripts can also transform message content at various points in the Cloverleaf engine. The user can write these scripts in an external text editor and import them to Cloverleaf, or use the built-in editor.

See [Introduction to the Script Editor](#).

- In Cloverleaf, inbound and outbound messages are archived to an external database through the ODBC drivers. Support is provided for reading and writing messages directly to and from Oracle and SQL Server databases. This is accomplished by SQL statements and stored procedures through JDBC.

See [ODBC command reference overview](#).

- Cloverleaf security server stores access control lists in an embedded Derby database. Whenever a user attempts to run an action, the host server queries the ACLs through RMI. This determines whether the user has permission to run that particular action. Users can be granted full read/write or read-only access to various areas.

See [Introduction to advanced security](#).

- As the Cloverleaf engine records statistics in shared memory, a separate monitor daemon process accesses shared memory to report these statistics to the user. The user can configure alerts to have the daemon send an email, pop up a window, or run a script when trigger conditions are met. These trigger conditions can be based on system statistics such as CPU and disk usage. They can also be based on Cloverleaf variables such as queue depth and protocol thread status. The monitor daemon communicates with the host server or directly with the Global Monitor through TCP/IP.

Multi-process and multi-thread design

The thread is a single strand that runs within a process, and is the basis for the configuration and monitoring abilities. The engine's ICL, Interthread Communications Library, maximizes thread communication, even across interprocess messaging or multiple process messaging. Intraprocess messaging, or single process messaging, takes advantage of a shared address space between threads in the same process, and runs faster.

Because intraprocess messaging shares the same process, it does not require data copying or a system request to deliver a message. It is often the fastest type of messaging available.

The engine is designed with three types of threads, each with a specific function:

- The protocol thread sends and receives protocol messages on a connection. It manages protocol connections to remote systems. This thread reads and writes data to and from devices, such as, TCP/IP

sockets, files, and SNA connections. All protocol threads are bi-directional. There is no limit to the number of protocol threads that can be defined within a process.

- The translation thread translates messages between several formats, and sends them to the protocol thread controlling the destination connection. There is only one translation thread per process. This thread transforms messages from the format of the sending system to the format of the receiving system.
- The command thread determines which protocol or translation thread has work to do, and schedules threads to run. This thread controls all other threads within a process, moving messages from thread to thread. There is only one command thread per process.

Tcl

Tool Command Language (TCL) is a non-proprietary, full-featured language that controls and extends applications. It is an embeddable and extensible scripting language that uses generic programming facilities, such as variables, loops, and procedures. These are useful for a variety of applications. Because Tcl is interpreted, code can be written on one platform and then moved to another without modification.

User Tcl modules are run at various places within the engine to permit users to customize message processing. Tcl 8.4 uses a bytecode compiler within core Tcl that provides runtime performance for the second and later invocations of Tcl procedures.

Custom Tcl extensions are provided for customization of messages within the engine through User Points of Control (UPoCs). Use these UPoCs to modify message content and message routing.

Boot-time customization

Boot-time customization is a method for processes to automatically start when a machine boots.

Note: This is not the default setting.

To configure this feature, go to `$HCIR00T/AutoStart.cfg`. This file contains the configuration instructions.

To configure, use these guidelines:

- `[site]` indicates in which site the auto-start process is located. Each `[site]` is followed by only one site name.
- `[process]` indicates the auto-start process name after the `[site]` line. `[process]` accepts a single process name, or multiple processes that are separated by a space. The process name can also be written in a new line under `[process]`.
- `#` comments the `[site]` and `[process]` line, if you do not require to auto-start those processes.

For example:

```
[site] sitename
[process] process1 process2
process3
```


Note: `AutoStart.cfg` can be modified as new processes are added to your configuration.

Monitoring and control

A varied set of tools is provided to monitor system performance, including:

- Event reporting and logging that reports when an event happens and records it in an event log.
- Operating margin monitoring with thresholding and alerts that monitor system capacity.
- Transaction statistics, such as the number of transactions, instances, and time spent.

The event reporting and system command/monitoring facilities enable a system that can operate:

- In a stand-alone mode, as host server and client.
- As part of a secure integration system combining disparate systems containing sensitive information into a communicating whole.
- As part of a larger network control scheme, using remote client configuration and monitoring capabilities.

The message configuration and protocol information can also be modified. Users can start, stop, redefine, add, drop, or control messages and their delivery to and from external systems.

Site monitoring

Cloverleaf users routinely rely on the Cloverleaf Monitor Daemon to monitor the health/status of interfaces. In today's distributed enterprise, users require a method/API to verify that the site daemons, `hcimonitor` and `hcilockmgr`, are functioning correctly.

To do this, there is a Restful API request that checks the status/health of `hcimonitor` and `hcilockmgr`.

This feature is enabled with CLAPI.

The API checks the current health of the two site-level daemons. The default behavior is to return the status on all Cloverleaf sites. To return the status on a single site, you can add a `sitename` parameter.

SiteDaemonHealth

POST	clapi/api/health			
Key	Type	Length	Required	Note
site	String	50	O	sitename

Example 1: This returns payload reports for the status of all sites.

`https://servername:port/clapi/api/root/sites/health`

Return:

```
[{ "lockmgrState": 1, "monitordState": 1, "siteName": "Site1" }, { "lockmgrState": 1, "monitordState": 1, "siteName": "Site2" }]
```

Example 2: This returns payload reports for the status of one site (prod).

`https://servername:port/clapi/api/root/sites/health/prod`

Return:

```
{ "lockmgrState": 1, "monitordState": 1, "siteName": "prod" }
```

Example

In this scenario:

- Cloverleaf is deployed in a Cloud environment.
- CLAPI is enabled with any dependencies.

To perform a site-level health check through CLAPI, you must configure a third-party application, Big Monitor, to poll Cloverleaf for health checks every few minutes.

Big Monitor calls the restful API to check the current status on the site-level daemons.

The return payload is parsed and the daemons are logged as being active.

Rebranding CIS and Security Server

Companies can resell the Cloverleaf software under their company name. To do this, you must make changes to the product name and service name. The installer checks for the existence of a `brand` directory in the root directory of the zip/tar file downloaded from the DLC.

If the directory exists, then the values in the `brand/product.txt` file are used to set the values of the variables `ProductName`, `CompanyName`, and `ServiceName`. Based on these values, the registry settings, program folder, and NT service is created with the rebranded name.

At the end of installation, the `CL_branding.pl` Perl script is run. This changes the Cloverleaf name with the rebranded name in several files that are visible to users during runtime.

Cloverleaf branding references files located in the runtime folder `%HCLIENT%\client\brand`. These files store the product name, company name, support information, icons, and flash screen. The sources of these files default to Cloverleaf.

For example, the default contents of `%HCLIENT%\client\brand\product.txt` is:

```
Infor Cloverleaf(R) Integration Services
Copyright (c) 2021 Infor. All rights reserved.
Infor
Infor Cloverleaf(R) Integration Services
IB
```

To change the software name that displays on the Windows uninstall dialog box, start-up menu, and Windows registries, you must create and modify `/installer_temp_directory/brand/product.txt`. To do this:

- 1 Before installation, create a `brand` folder under `/installer_temp_directory/`.
- 2 Copy all rebranding files, including icons and flash screen, to the folder.
- 3 Start the installation.

The installer reads the `product.txt` file to get the product name and company name. It then generates the correct NT service name and start menu icon names.

The installer copies all content in the `brand` folder to the runtime directory at `%HCLIENT%\client\brand`. This is where the IDE and engine runtime retrieves this information to display properly.

- 4 After installation, you can modify (rebrand) the product name, company name on the GUI's title, and log information, in `%HCLIENT%\client\brand/`. You must restart the server and re-launch the GUI for the changes to take effect.

These are the files that can be rebranded:

- `product.txt`
Defines product name and company name.
- `brand.txt`

Defines the support information.

- `brand-splash.gif`
The splash-screen image that is used when launching the GUI.
- `brand-16x16.png`
The icon that is used at the top left of the GUI.

Revision control

All load and save operations create a lock and revision history log file. This log file is in:

- Windows: `%HCISITEDIR%\revisions\Revisions.log`
- UNIX: `$HCISITEDIR/revisions/Revisions.log`

Revision history for configuration files is located in the same directory structure as the site directory. Files are stored using the same name, plus an extension made up of the file's creation date.

The number of revisions is not limited, so administrators should periodically remove old copies.

Configuring the network

Conceptualizing network interfaces can shorten the implementation time dramatically. This section summarizes the processes involved in network configuration, and assumes that all systems involved are functional.

Before building a site, assemble this information:

- The transactions that are used to and from the system.
- The platform make and model and operating system.
- The vendor. For example, SMS.
- The system type. For example, Accounting.
- The communication protocol. For example, LU 6.2 Token.
- The message format. For example, ADT, Person Records, or Charges.
- The level of security. For example, No security, basic, or advanced.
- Message action. For example, raw or translated.

Implementation approaches

You can configure a network in one of three ways.

From the connection: "Structure with direction" provides an overall view of the network system and its needs.

- 1 Configure the host connections.
- 2 Configure the translation lookup tables.
- 3 Configure the FRL definitions.
- 4 Configure any non-fixed record layouts, such as XML, VRL, or HRL.
- 5 Configure any record layouts that use variants, such as HL7, UN/EDIFACT, NCPDP, or X12.
- 6 Configure the record translation specifications.
- 7 Edit the network connection and route configurations to supply any missing information. Configure the transaction routes between the hosts. This step shows the sources and destinations for message transactions.

From the record layout: "Details first" is a logical way to enter configuration information in the system database. This does not show an overview of the network until the entire configuration is complete.

- 1 Configure the FRL definitions.
- 2 Configure any non-fixed record layouts, such as XML, VRL, or HRL.
- 3 Configure any record layouts that use variants, such as HL7, UN/EDIFACT, NCPDP, or X12.

- 4 Configure the record translation specifications.
- 5 Configure the translation lookup tables.
- 6 Configure the message translation specifications.
- 7 Configure the host connections.
- 8 Configure the network connections, threads, and routes.

From the network: "Structure first" shows an overview of the entire network before details are added. The disadvantage of this approach is that you cannot configure network objects, such as threads and routes, until after configuring message layouts and translations.

- 1 Configure the network connections, threads, and routes.
- 2 Configure the FRL definitions.
- 3 Configure any non-fixed record layouts, such as XML, VRL, or HRL.
- 4 Configure any record layouts that use variants, such as HL7, UN/EDIFACT, NCPDP, or X12.
- 5 Configure the record translation specifications.
- 6 Configure the translation lookup tables.
- 7 Configure the message translation specifications.

Before implementation

An interface communication system is complex to implement. Before beginning an implementation, outline the necessary steps.

Note: Every site is unique. Consult with System Implementation team members when constructing a plan.

Task overview:

- 1 Outline the system requirements.
- 2 Appoint the system integration team.
- 3 Assign team members to specific interface requirements. Assign by system, data, or vendor needs.
- 4 Complete the interface request form for each vendor or system.
- 5 Evaluate the interface and outside system requirements with your team.
- 6 Develop the scope, work plan, and budget.
- 7 Complete the specs form.
- 8 Develop a test and migration plan.
- 9 Construct the interface.
- 10 Test the interface.
- 11 Implement the interface.

Internal documentation is important throughout the interface cycle. Customize the documentation for each project, so that scope and expectations are always clearly defined.

System configuration

Use the Network Configurator to sketch an overall view of the network system and its needs, and to finish message configuration details.

To configure "From the Connection":

- 1** Open the Network Configurator from the IDE.
- 2** Click the connection icon to open the connection menu. Select **Thread Properties** to configure the host connections.
- 3** To configure the transaction routes between the hosts, select **View > Show All Routes** on the Network Configurator. This step gives an overview of the network, showing the sources and destinations for message transactions. To set up routes, configure routes as static routes with raw data translations.
Not all routes are static with raw data, but configuring routes as static shows an overview of the routes before adding route detail.
- 4** Configure the translation lookup tables.
- 5** Configure the FRL definitions.
- 6** Configure any non-fixed record layouts, such as XML, VRL, or HRL.
- 7** Configure any record layouts that use variants, such as HL7, UN/EDIFACT, NCPDP, or X12.
- 8** Configure the record translation specifications.
- 9** Edit existing routes and translations.

Command line license help tool

The command line license help tool is a streamlined method to gather human-readable license information from a user's installations.

The license script assembles the data elements that are required to license the client hardware and additional server environment metrics. This information is used by Support to troubleshoot issues.

This feature is installed by default.

User interface: Command line interface

The license script is run from a shell session after running the `setroot` command. This script accepts these parameters:

- The output file name as a command line argument. The report is written to the file name. If no file name is specified, then the output is written to `stdout`.
- An optional license file name for the license file that is included in the report. If no license file is supplied, then the report uses the current root.
- A switch that produces the license report. See *Server interface: License report* (below).
- An optional switch that produces the license audit report. See [License help audit report](#).
- An optional switch that produces the delimited audit report. See [License help delimited audit report](#).
- A parameter that reports/groups the audit report by site.

Server interface: License report

The command line license script assembles these server environment settings, and the required elements for the specific product being licensed.

- Platform.
- Product release, using `hciengine -v`.
- CPU cores, using `hciengine -v`.
- Current license information, using `hcilicstatus`.
- Security level, using `hcilicstatus`.

If no output is received from `hciengine -v` due to an earlier version, then the script runs an included script. This script assembles the current product release and available CPU cores.

This table lists the products and their license feature keys:

Product Name	HostID	License Feature Keys
Cloverleaf Integration Services		<ul style="list-style-type: none"> cl-pkg-cl cl-aom-webservices cl-aom-site-doc cl-mm-master cl-cp-master cl-aom-caa-ion cl-mm-ncpdpall
Cloverleaf Integration Services threads		<ul style="list-style-type: none"> cl-intfc-thread-## cl-intfc-master
Cloverleaf Security Server		cl-aom-sec-advanced
Cloverleaf Secure Courier		cl-aom-csc-server-##
Cloverleaf Global Monitor		cl-aom-gmon
Cloverleaf Secure Messenger		<ul style="list-style-type: none"> cl-aom-ssl cl-aom-ftps cl-aom-sftp
Cloverleaf Application Adapter – Web Services		<ul style="list-style-type: none"> cl-aom-caa-ws cl-aom-caa-direct
Cloverleaf Message Warehouse		cl-aom-msgwarehouse
Cloverleaf ODBC Data Integrator		<ul style="list-style-type: none"> cl-aom-odbc-mw cl-aom-odbc-tcl
Cloverleaf Tcl extensions		cl-aom-odbc-tcl
Infor FHIR Bridge		cl-aom-IFB
Cloverleaf IHE		cl-aom-IHE
Infor Clinical Bridge		cl-aom-ICB

Script summarizes the features the client has licensed in a format similar to the Infor Support Portal or Concierge license request page.

This report contains the correct work flow for the Infor Support Portal or Concierge. That is, **Resources > Request a Software Key**.

The report gives the user explicit instructions on what fields and data elements to fill out on the license key request form.

License help audit report

The command line interface has an optional switch that produces the license audit report.

This report contains these sections:

- Core Count (logical):

Reports the number of CPU cores that are available on the server. For example:

In Windows:

```
wmic cpu get NumberOfCores, NumberOfLogicalProcessors/Format:List
```

In Linux:

```
Linux: grep -c ^processor /proc/cpuinfo
```

- Modules that are used in NetConfigs across sites:

Summary report on the number of modules that are used in NetConfigs across sites in the root. For example:

```
15 threads using secure TCP/IP encapsulated
7 threads using Java Driver
8 threads using Web Services Adapter
65 threads using non-secure TCP/IP encapsulated
10 threads using secure fileset/FTP
22 threads using non-secure fileset/FTP
```

- Courier Count that is used:

Count of the number of courier configurations.

For example:

```
ls -l */*.cfg.public | wc -l
```

License help delimited audit report

The command line interface has an optional switch that produces the delimited license audit report.

The delimited audit report contains the same information as the audit report, but in a delimited CSV format that can be imported into external software.

One line is listed per site. The default value for all thread types is "0". All strings are encased in double quotes. This report contains these fields:

- Cloverleaf
 - Platform
 - CPU cores (logical)
 - Site name

- Thread number
 - Threads: File
 - Threads: Fileset/local
 - Threads: Fileset FTPS
 - Threads: Fileset SFTP
 - Threads: non-secure TCP/IP
 - Threads: UPoC
 - Threads: secure PDL TCP/IP
 - Threads: non-secure PDL TCP/IP
 - Threads: Database protocol
 -
 - Site number
 - Secure Courier
- Number of couriers

License help information storage

The `hcilicelp` script accepts a file name as a command line argument. A report is written to the file name. If no file name is specified, then the output is written to `stdout`.

The license report contains these columns:

- Product:
See *Server interface: License report* in [Command line license help tool](#).
- User Count:
Thread count and courier count, if available.
- Release:
Currently installed version.
- Request key:
Default is "Yes."

This is an example of a sample report:

Platform: Linux Redhat 7.3
 Product: Cloverleaf 6.2.1
 CPU cores: 4
 Security level: None

Product	User Count	Release	Request Key
Cloverleaf Integration Services	N/A	6.2.1	Yes
Cloverleaf Integration Services Threads	18	6.2.1	Yes
Cloverleaf Secure Messenger	N/A	6.2.1	Yes

License help command usage

The command line license help tool is a streamlined method to gather human-readable license information from their installations.

Command usage is:

```
hcilichelp [-i license filename] [-o report filename]
           [-c CSC Root] [-a] [-s] [-d]
```

- `-i license filename` is an optional license file name for the license file used for the report. If no license file is supplied, then the report uses the current root.
- `-c report filename` is the output file name as a command line argument. The report is written to the file name. If no file name is specified, then the output is written to STDOUT.
- `-c CSC Root` reports the CSC license and courier information that are based on `CSC Root`.
- `-a` generates the audit report.
- `-s` groups the audit report by site.
- `-d` delimits the audit report to CSV. See [Delimited report](#).

The `hcilichelp` command gathers this information:

- Machine information:
 - OS version
 - CPU logical core
- CIS version:

If the tool works on an earlier CIS version, it gets the major version from `$HCIR00T`. The tool gets the minor version from `$HCIR00T/CISPatchLevel`.
- CSC version:

The tool gets the CSC version from `$CSCR00T/server/version`.
- Courier count:

The tool gets the courier number by counting the folders under `$CSCR00T/sever/lws/registered`. Each folder is a registered courier.
- Default license file:

When `-i` is not given, the tool reads the CIS default license file at `$HCIR00T/vers/license.dat`.

When `-c` is given, the tool reads the default CSC license file at `$CSCR00T/server/vers/license.dat`.

Feature table

The feature table records the relationship between feature keys in the license file and the product name for the user to request.

See [Command line license help tool](#).

For products that map to multiple keys, as long as there is one key among the corresponding keys that exists in the license file, the product name is shown in the license report. It does not require that all the keys must be in the license for a product.

Delimited report

The delimited audit report contains the same information as the audit report in a delimited CSV format. This format can be imported into external software.

The `-d` option implies `-s`, as it always generates the site level detailed information.

The first line is the header, and each following line is per site.

The report uses a comma as a separator. All strings are enclosed in double quotes. The default value for all thread types is "0" (zero).

License help examples

In these scenarios, An Interface Developer is generating a license report:.

Generating a license report on the same machine

To install a new Cloverleaf version, you must confirm your current licensed instance and request a new license file for the new Cloverleaf version.

Then, open a shell session in the Cloverleaf server and set the correct root using `setroot`. After this, run this command:

```
hcilichelp -o /home/hci/license_report.txt
```

The license report is saved to the file name listed in the command. You can now retrieve and view the report.

Submit the license request through the Infor Support.

Generating a license report on a new machine

To install a new Cloverleaf version on new hardware, you must confirm your current licensed features and request a new license file for the new hardware.

Then, copy the `license.dat` file to the new machine in the `hci` home directory using the name `license.old`.

Open a shell session in the Cloverleaf server and set the correct root using `setroot`. Then, run this command:

```
hcilichelp -i /home/hci/license.old -o /home/hci/license_report.txt
```

The license report is saved to the file name listed in the command. You can now retrieve and view the report and submit the license request to Infor Support.

Directory hierarchy

When you install the system in Windows, its files are loaded into `cloverleaf`. The directory hierarchy is structured so that each `HCISITEDIR` site contains customizable directories.

When you install the system in UNIX, its files are loaded into `/cloverleaf`. The directory hierarchy is structured so that each `HCISITEDIR` site contains customizable directories.

When you install the system in Windows, its files are loaded into `\cloverleaf`. The directory hierarchy is structured so that each `HCISITEDIR` site contains customizable directories.

integrator subdirectories

Subdirectory	Contents
Alerts	Sample alerts file.
AppDefaults	X resource definitions. These control visual aspects of the GUIs, such as color, fonts, and icons. They are not user-configurable.
archiving	Contains <code>create_template.sql</code> and <code>insert_template.sql</code> .
bin	Root level binaries, program scripts, and Raima (database) binaries.
box	Contains the configured BOX files.
bitmaps	Program icon bitmaps. This is not user-configurable.
CAA	Contains the files for CAA-ION, CAA-Direct, and CAA-WS.
clgui	Java-based components. Contains the <code>hci</code> wrapper scripts. The subdirectories are all Java components such as the Java Runtime Environment and library classes.
client	Client configuration directory; contains <code>client.ini</code> and cached SSL credentials.

Subdirectory	Contents
contrib	Unsupported scripts that are used by implementors on site.
dicom	Contains the dicom configuration file.
docs	Contains information that is related to the system Java UPoC API.
eoalias	Engine output configuration files. When primary-level files are unalterable, Root-level files that are created with EO Configurator are available to all sites.
formats	Format files for DBSCHEMA, deprecated, EDIFACT, HL7, HPRIM, LDL, NCPDP, NCPDPFAB, NCPDPSCRIPT, X12, and XML.
helloworld	The sample site included with the installation.
JavaDriverSamples	Contains the Java driver sample sites.
java_uccs	Recommended for keeping root-level Java UPoC *.class files.
lib	Library files for public domain tools, perl libraries, and for ODBC middleware clients.
pdls	Common PDL (Programmable Driver Language) programs.
perldocs	(Windows only): Win32 Perl extension documentation.
productinfo	Default root setting. Contains software version and licensing information. The default root is used when the <code>setroot</code> command is run without any arguments.
sbin	Root level binaries and program scripts that can be used before HCIRoot is set. This contains mainly <code>hcisetenv</code> , which is used by <code>setroot</code> .
security	Security server configuration directory; contains <code>security.ini</code> . This folder exists only when security server is installed.
server	Host server configuration directory; contains <code>server.ini</code> .

Subdirectory	Contents
shared_files	Contains files managed by File Transfer. When using File Transfer, users of client-only installations who are remotely connecting to the host server can manipulate files on the remote machine. This can be performed by copying files to a fixed directory on the server.
sitename	Site information that includes configuration files, database, and Tcl procedures.
siteProto	Site prototype that contains the default directory structure for site creation.
startup	(Windows only): Contains the script for starting <code>hci-clguiservice</code> .
Tables	Non-site-specific lookup tables. These can be copied over to <code>\$HCISITEDIR/Tables</code> for customization.
tcl	Tcl extension language; distributed programming package; extended Tcl, Tk files, and scripts; and a library of Tcl autoload procedures. Tcl shared libraries are located in <code>/tcl/lib</code> on UNIX platforms and in <code>bin</code> on Windows.
tclprocs	Contains provided Tcl procedures. This contains a code set template Tcl procedure file that gives an example of code to use to query a lookup table for a key.
templates	Contains the NetConfig template files.
uninstall_integration_services	Contains the uninstaller files.
usercmds	User-created commands.
vers	Version and licensing information.
version	When a user adds a configuration into Version Control, a <code>version</code> folder is created in the current site to store the configuration. The root-level <code>version</code> folder is used for the BOX Manager tool.
web	Files that are related to the HTTP server plug-in. This includes perl scripts. These can be plugged into a web server. Then, a web server interface into the system is created by permitting it to route web requests by File or TCP/IP threads. <code>CL_HTTPServer.ini</code> is included as a configuration file for this interface. This also contains sample HTML files.

Subdirectory	Contents
xslt	XSLT configuration files. This includes the XSLT validation files for HL7 CDA 2.0 and HITSP_C32 (2.5 version). These files are found in the root level XSLT folder (\$HCIR00T/xslt). They can be used by <code>hcixslttest</code> or the engine without any changes. They can also be copied and modified to the site level (\$HCISITEDIR/xslt or \$HCIMASTERSITEDIR/xslt) for special use.

\$HCISITEDIR site-specific subdirectories

Subdirectory	Contents
Alerts	alrt files that are used to monitor the system.
AppDefaults	Site-specific X resource definitions.
archiving	Contains site-level templates. These override the root-level templates.
eoalias	Site-specific engine output configuration files.
exec	Contains engine and daemon script directories, with subdirectories for storing data generated by the engine.
formats	Site specific user-defined format files for DBSCHEMA, EDIFACT, FRL, HL7, HPRIM, HRL, LDL, NCPDP, NCPDPFAB, NCPDPSCRIPT, VRL, X12, and XML.
javadriver	Contains Java driver files when a java protocol is selected in Network Configurator.
java_uccs	Recommended for keeping site-level Java UPoC *.class files.
lock	Location of files that are locked by Lock Manager in a live site. If a file named disabled exists in this directory, then locking is not performed.
notes	Contains note files for notes written in Network Configurator.
pdls	Site-specific PDL (Programmable Driver Language) programs.

Subdirectory	Contents
revisions	Location of files that are modified from original configuration. The file name format is <code><file name>.<date>.<time></code> . These files are not automatically cleaned. The administrator should periodically clean up earlier revisions.
smatdb	Contains criteria definitions for the SMAT database.
Tables	User-created lookup tables.
tcl procs	Site-specific Tcl procedures.
version	When a user adds a configuration into Version Control, a <code>version</code> folder is created in the current site to store the configuration. The root-level <code>version</code> folder is used for the BOX Manager tool.
views	Views and alternate views, maintained and used in Network Configurator or Network Monitor.
Xlate	Translation configuration files.
xslt	XSLT configuration files. This includes the XSLT validation files for HL7 CDA 2.0 and HITSP_C32 (2.5 version). These files are found in the root level XSLT folder (<code>\$HCIRoot/xslt</code>). They can be used by <code>hcixslttest</code> or the engine without any changes. They can also be copied and modified to the site level, <code>\$HCISITEDIR/xslt</code> or <code>\$HCIMASTERSITEDIR/xslt</code> , for special use.

\$HCISITEDIR exec subdirectories

This table lists the `$HCISITEDIR exec` subdirectories:

Subdirectory	Contents
databases	This contains error and recovery database files. To clear the error database, use the Database Administration tool or the <code>hcidbdump</code> command-line utility. Note: Do not delete files from this directory!
errors	Engine output file and core dumps are copied here when the engine ceases operation abnormally, such as during a power failure. Core dumps are on UNIX platforms only. Earlier files can be deleted from this directory.

Subdirectory	Contents
hcilockmgr	<p>Process script and process ID directory for the lock manager.</p> <p>Note: Do not delete files from this directory!</p>
hcimonitor	<p>Process script and process ID directory for the monitor daemon. This contains:</p> <ul style="list-style-type: none"> cmd_port: Command port number used by other processes to access the monitor daemon's server socket port. They can then connect and receive status updates or send commands, such as cycle. hcimonitor.err: Errors and warnings are written here. hcimonitor.log: General monitor daemon log file for all log messages; similar to the engine output log file. Currently this is only controlled from the command line <code>-de</code> flag. No eoconfig exists for the Monitor daemon. hcimonitor.log.old: Previous log file, if one exists when the monitor daemon is started. <p>Note: Do not delete files from this directory!</p>
hcimsgarchive	<p>Contains the saved archive files. This is configured on the Site Preference's Message Archiving tab. Message logs are cycled when they get to the specified size. Cycling a save file causes the current file to close, renames it to <code>old_name.old</code>, and opens a new save file.</p>
monitorShmemFile	<p>This is an MSI shared memory file that is used as a key by various programs. These programs locate the shared memory segments containing the engine runtime statistics. Specifically, it is the file used in an <code>ftok</code> invocation which returns a key that is used in the memory attach operations. This file contains no user-usable data. It is a pipe used by the engine to relay thread state and status changes to MonitorD in an event-driven manner.</p> <p>In UNIX, this is a named pipe used for communication between <code>hcimonitor</code> and all the running engine processes in the site.</p> <p>In Windows, it is a memory mapped file used to store statistics.</p> <p>Note: Do not remove this file when processes are running!</p>

Subdirectory	Contents
processes	Subdirectories for each engine process. Each subdirectory is referred to as a "Process Script Directory." This is the relative path name for files that are created by an engine process, such as process logs and SMAT files.

Cloverleaf Integration Services IDE

The IDE is a group of tools that are used to create, configure, and monitor sites.

Note: Due to differences in tool options and the amount of configuration information, some tools might not display well. To correct this, you can resize the IDE, close the Site Manager, or close the Launch Bar.

This table lists the IDE components:

Component	Description
Title bar	Includes the version number, the name of the active tool, and the name of the current file. If the current file has been modified, but not saved, then its name has an * (asterisk) appended to it.
Menu bar	Includes the menus and menu options that apply to the IDE in general. It also contains the menus and options that are specific to the currently active tool.
Toolbar	The tools on the toolbar are supplied by the active tool. Any changes are tool-specific.
Status bar	Located at the bottom, this shows information on the current site and version. Click the logo to see client, support, server, and license information. Optionally, you can add a site description to the status bar.
Launch bar	Contains the runtime, configuration, and testing tools.
Site Manager	<p>Provides access to all files within the currently connected site. Right-click a node to open a menu of version control options.</p> <p>All configuration files can be opened from the Site Manager as read-only by selecting Open as read-only. To return to edit mode, select File > Change to Edit Mode.</p> <p>When there are multiple selections of Site Manager tree nodes, this option is not displayed.</p>
Tool panel	Shows the running tools. If there is more than one running tool, then each tool has its own tab.

Component	Description
Tool tabs	Tabs include a right-click menu that contains closing and saving options.
Split bars	These can be dragged to minimize/maximize the panel.

Site Manager

The Site Manager shows the contents of the current site. It shows folders for all the file types within a site and lists the contents of those folders. You can double-click the folder content to open them in a configuration tool.

When the IDE is launched, the Site Manager is not shown if it was hidden when the IDE was shut down the previous time. In this case, the IDE does not initialize the content of the Site Manager. This also applies to when the IDE changes sites. The Site Manager is initialized if it has not yet been initialized. After initialization, the IDE does not reinitialize unless the site is refreshed.

When a file is saved, only the folder to which this file belongs is reloaded, instead of the entire site.

Note: All tables must have the `.tbl` extension to be viewed in the Site Manager and menu lists within Translation Configurator. If these tables are created from within the GUI, then they already have the `.tbl` extension. Sites that are brought forward from previous versions that do not use the `.tbl` extension must have the table files renamed. The names must include the extension and the references must be updated in the translation that uses the table.

The Site Manager has a **Refresh** feature to update the tree contents. This is to handle files that may have been added to a site outside of the IDE.

The `*.tcl` nodes contain child nodes for the Tcl procs within that file. These nodes are expandable to show the procedures that they contain. Double-clicking a procedure node and a `.tcl` file node open that procedure and file.

Search functionality

Specify the target string in the search field at the top of the Site Manager and click **Search** (binoculars).

Searches are case insensitive.

Searching starts at the currently selected item. If there is no selected item, then the search starts at the root node. When an item is found, it is highlighted. If it is hidden in the view, then the its parent node is opened to show where it is located.

If there is no search match, then a message box opens stating that no item was found.

The search starts again from the top when the last item is found and **Search** is clicked again.

Items are sorted in ascending order when the Site Manager is first opened. You can switch between ascending and descending order by clicking **Sort**. The sorting order stays the same when **Refresh** is clicked or a new item is added.

As long as the view is open, the sorting order persists. When the IDE is closed, the sorting order reverts to ascending.

Open as read-only

To change a tool to read-only, select **File > Change to Read-Only Mode**

If a tool is already open, then this option is disabled; otherwise, it is enabled. When there are multiple selections of Site Manager tree nodes, this option is not displayed.

When read-only is enabled, `Read-Only` is displayed on the status bar.

To disable read-only, select **File > Change to Edit Mode**.

Site Init dialog box

The **Site Init** dialog box is accessed at **Start > All Programs > Infor Cloverleaf Integration Suite > Tools**.

Option	Description
Select a root directory	Select the root directory where Cloverleaf is installed.
Select a location for sites outside HCIRoot (Optional)	Select a target path where the new site is saved. The symbolic link <code>\$HCIRoot/newsite</code> is created, linking to the <code>path/newsite</code> after site creation. This is only available in NTFS on Windows. The target path is R/W to the user.
Select a Site as Template	Select a site from the list to use as a template. With this feature, you can define a template site instead of modifying <code>siteProto</code> .

Command line interaction with security

Most command lines are designed as local commands and have no interaction with the Security Server. The best practice recommendation is to use the related CLAPI, CLWizard function, or GUI tool.

The command line `hcisiteinit` is designed as a local command line to create a site that is installed on the Host Server.

For example, the `hcisiteinit` command line only creates a site on the Host Server under Basic/Advanced Security mode. The Site Init GUI tool, CLAPI, or CLWizard have log-in features for user authentication

(Basic/Advanced Security). They also check the user permissions to create a site (Advanced Security) and uploading the created site to Security Server (Advanced Security).

The Site Init GUI is a client-side tool that interacts with the Host Server and calls `hcisiteinit` to create a site on the Host Server.

The Site Init GUI performs these tasks under Basic/Advanced Security mode:

- User log-ins to the selected site. It sets up the TLS connection to the Host Server and checks whether the user has run permission on the Site Init tool.
- Requests the Host Server to create a site with `hcisiteinit` and verifying if the user has run permission on `hcisiteinit`.
- Updates `server.ini`.
- Connects with the Security Server to upload the created site.
- Audits who and when logs in to the Site Init GUI.

Creating a site in Windows

In Windows, sites are created using the **Site Init** dialog box.

- 1 Select **Start > All Programs > Infor Cloverleaf Integration Suite > Tools > Site Init**.
- 2 Specify a new site name in the name field.
- 3 If required, then select a root directory by clicking **Select a root directory**.
The default directory is already selected.
- 4 Select a site location at **Select a location for sites outside HCIROOT (Optional)**.
If left blank, then the default location is in the `integrator` folder.
- 5 You can optionally select a site from the **Select a Site as Template**.
- 6 Select the check boxes as required.
- 7 Click **OK**.
- 8 To access the new site, in the IDE select **Server > Change** to open the **Server/Environment** dialog box.
- 9 Locate the new site and click **Apply**. The IDE now reflects the site change.

Creating a site in UNIX

To create a site in UNIX:

- 1 In UNIX, open a Shell Window.
- 2 `setroot` to the appropriate root.
- 3 Run `hcisiteinit`.
- 4 Perform a `setsite newsite` operation.
- 5 Verify using `showroot`.
- 6 Specify `hciaccess &` and then select **Enter** to open a new dialog box.

- 7 Use **Server > Change** on the menu bar to access the new site.

Symbolic links

Using symbolic links, you can move SMAT files to locations outside the Cloverleaf root directory. These files can be opened from any location.

Symbolic links are only supported for linking an entire site to another directory outside `$HCIRoot`. The symbolic linking of `$HCISITEDIR` sub-directories is not supported.

On UNIX and Windows installations, the **Site Init** tool has a **Select a location for sites outside HCIRoot (Optional)** option. This is for a symbolic link. A symbolic link creates a site from a user-specified location at `$HCIRoot/newsite` and creates a link to the new site at `path/newsite`. This location can be outside `$HCIRoot`.

This is only available in NTFS on Windows. The target path is R/W to the user.

Clicking the folder button opens the **Export to Directory** dialog box, where you select the location.

If a site is configured with a symbol link, then the link is shown when the site is selected on the **General** tab of the **Server Administration** dialog box.

Command-line usage is also available. See [hcisiteinit](#).

Symbolic-linked SMAT and SMAT database files outside of HCIRoot

You can move SMAT files to locations outside the Cloverleaf root directory. These files can be opened from any location.

See [Loading SMAT history outside the SMAT folder](#).

Creating a symbolic link to a moved folder

This is an example of how SMAT/SMAT database can load files under a process folder that is a symbolic link.

The symbolic link points to a folder outside of `hciroot`.

- 1 In the Site Init tool, create a new site named **symbolicprocess**.
- 2 Move the processes folder in `C:\cloverleaf\cis6.2\integrator\symbolicprocess\exec` to `C:\testprocesses` (outside of `%HCIRoot`).
- 3 Create a symbolic link using the command line to point to the moved folder.
 Windows: `mklink /D C:\cloverleaf\cis6.2\integrator\symbolicprocess\exec\processes C:\testprocesses\processes`
 Linux: `ln -s /opt/testprocesses/processes/opt/cloverleaf/cis6.1/integrator/symbolicprocess/exec/processes`
- 4 Open the IDE and change to the `symbolicprocess` site.

The `smatdb` files in the `processes` folder are shown in the SMAT Database File List pane. The `smatdb` files can also be opened in Network Monitor.

Creating a symbolic link for `smatdb` files that are outside `HCIRoot`

You can create a symbolic link for `smatdb` files that are outside `HCIRoot` when using the GUI to search.

`hciuser` has the "Create symbolic links" privilege on Windows.

- If there is not an `hciuser` before installing CIS on Windows, then this issue does not apply, because the `hciuser` is generated by CIS.
- If there is an `hciuser` that was created on Windows by an earlier CIS version, then you must do the steps below after installing CIS.

SMAT database > SMAT database cycling.

- 1 Run the `gpedit.msc` command.
- 2 Add `hciuser` to the "Create symbolic links" policy under **Local Computer Policy > Computer Configuration > Windows Settings > Security Settings > Local Policies > User Rights Assignment**
- 3 In the Policy panel, click **Create symbolic links**. This opens the **Create symbolic links Properties** dialog box.
- 4 Click **Add User or Group** to add `hciuser`.
- 5 Reboot the operating system.

Loading SMAT history outside the SMAT folder

You can save SMAT history files outside the site folder by creating a link path to a specific folder/disc. You can then subdivide `smatdb` files within the `smatHistory` base on thread names.

These options are available from **Options > Site Preferences > SMAT > History Options**. These options are based on the site level. That is, the configuration is implemented on all processes. Processes within a site save the history database files to the link path.

- **SmatHistory Link Path**

When this is selected, specify the path to link in the field. The default value is null.

When you specify the actual path, the engine creates this link when it starts. For example, `-s ACTUAL_PATH $HCISITEDIR/exex/processes/process_name/smatHistory`.

- **SmatHistory saved base on Thread Name**

When this is selected, SMAT saves the `smatHistory` database files based on thread names. The default value is "0."

When this option is selected, sub-folders using the thread name are created within the `smatHistory` folder. History `smatdb` files are saved into these sub-folders. This is the meaning of saving `smatdb` files based on the thread name.

Note: When the **Select a location for sites outside HCIROOT (Optional)** option is used on the **Site Init** dialog box, and multiple processes are saved to the same linked directory, do not use the **Maximum** options in the SMAT History panel. See [Symbolic links](#).

Because the GUI is limited regarding SMAT searching, support is only for making the link for the `smatHistory` folder to a specific path. In this way, `smatHistory` can be subdivided base on thread names.

- 1 In **Options > Site Preferences > SMAT > History Options** set these conditions:
 - **Delete the oldest SMAT inbound/outbound history file when history file's number for a thread exceeds number files** to 1.
 - **Delete the oldest SMAT history file when history file's total size for a process exceeds number KB** to 2.
 - **Delete SMAT history file for a process when it is older than number days** to 3.
- 2 Clear **SmatHistory saved base on Thread Name** first, and then cycle. The `smatdb` files generate.
- 3 Select the check box, restart the engine, and then cycle. The `smatdb` files are saved into a sub-folder.

Using SMAT/SMAT database when the site is a symbolic link

In this example, the SMAT/ SMAT DB works if the site is a symbolic link pointing to a folder outside `hciroot`.

- 1 In the Site Init tool, create a new site named **symbolicsite**.
- 2 Move the `symbolicsite` folder in `%HCIROOT` to `C:\testsite` (outside of `%HCIROOT`).
- 3 Create a symbolic link using the command line to point to the moved folder.
 - Windows: `mklink /D C:\cloverleaf\cis6.2\integrator\symbolicsite C:\testsite\symbolicsite`
 - Linux: You can create a new symbolic link site using the Site Init tool, in the **Select a location for sites outside HCIROOT (Optional):** field.
- 4 Open the IDE and change to the `symbolicsite` site.
The `smatdb` files in the `site` folder are shown in the SMAT Database File List pane. They can also be opened in Network Monitor.

Command line usage for sites

This table lists site commands:

Command	Output
<code>showroot</code> (UNIX/Windows)	Displays your current root and site.
<code>cd \$HCIROOT</code> (UNIX)	Moves to your current root.
<code>cd %hciroot%</code> (Windows)	
<code>cd \$HCISITEDIR</code> (UNIX)	Moves to your current site.
<code>%hcisitedir%</code> (Windows)	

Command	Output
<code>hcisiteinit</code>	Creates a new site within the currently-selected root.
<code>hcisiteinit <i>newsitename</i></code>	The specified site cannot already exist. The standard set of directories are created within the new site.
<code>hciguisiteinit</code>	<p>This utility can only be used if the client software is loaded.</p> <p>For Windows clients on Windows or UNIX hosts, you must use the Site Init tool.</p> <p>For a UNIX client with UNIX hosts, use a shell window and the command <code>hciguisiteinit</code> or <code>hcisiteinit</code>.</p>
<code>setroot</code>	Selects a system release root, and possibly a site, in which to operate.
<code>setroot [rootdir [site]]</code>	<p>The specified <code>rootdir</code> must be a valid system release root. The site, if specified, must be the name of a site within the selected root.</p> <p>If no root is specified, then the contents of this file is used to find the name of the default root: <code>/cloverleaf/cis6.1/integrator/productinfo/default</code>.</p> <p>If no site is specified and the selected root's <code>rootInfo</code> file has a <code>site=</code> entry, then that site is automatically selected.</p>
<code>setsite</code>	Selects a site directory within the currently selected root.
<code>setsite <i>site</i></code>	The selected site must be a valid site within the current root.
<code>showroot</code>	Displays the current settings from previous <code>setroot</code> and <code>setsite</code> commands.

Version control

Note: This feature requires basic or advanced security to be enabled.

Using Cloverleaf source and version control, users can check files in and out. It also prompts notifications when a file is checked out. This prevents editing of the same configuration file by multiple users.

Check-out verifies whether the file is checked out by other users. If so, then an error message is displayed when **Save** is clicked.

Note: Files that have not been checked into version control use the current file locking mechanism.

Users can add any configuration into version control. The version folder is created in the current site or \$HCIR00T to store the configuration's version information.

Configuration files consist of those that are accessible through the IDE or other editors. Examples include NetConfig, alerts files, Engine Output alias, formats variants, Tables, Tcl Scripts, xlate, and xlts.

Each configuration tool has the version control actions on the **Edit** menu, along with the shortcut keys. **Edit** menu options are also available on the main toolbar.

All version control actions are only available for the current configuration.

This feature also includes BOX version and revision control. For the BOX Manager, a version folder is added to the root level.

For information on the command line usage, see [hcversion](#).

For a configuration, the administrator can remove the version records or unlock the checked out status. This is accessed on the **Server Administration > Version Control** tab using **Remove** and **Unlock**.

Version folders and database

Each version of a configuration file is stored in an individual folder.

The site level folder is located at \$HCISITE\version. This stores every checked-in configuration file, in the format \$HCISITE/version/file name with relative path/version number.

For example: C:\cloverleaf\cis6.2\integrator\helloworld\version\formats\hl7\2.1\test\1.1.

The root level folder is located at \$HCIR00T\version, in the format \$HCIR00T/version/file name with relative path/version number .

For example: C:\cloverleaf\cis6.2\integrator\version\box\test\1.1.

The version record database is a SQLite database. Each site has its own version database. This stores the operation and history version information.

For site level configurations, version records are stored in \$HCISITE\version\version.db.

For root level configurations, version records are stored in \$HCIR00T\version\version.db.

Site Manager context menu

The Site Manager tree nodes show the configuration file name with the latest version number.

The file tooltip contain version details at a glance, including Full Name, User, Operation, and Time.

A right-click context menu is available for configuration and folder nodes.

This table shows the available options:

Option	Description
Check In	<p>Adds the raw configuration into the version control system, or adds the current modification as a new version.</p> <p>This copies and saves the configuration file with the version number. Comments can be added at check-in.</p> <p>To add a new entry into the version control system or add a new version for the existing entry, you can use Check In.</p> <p>For a new configuration file, this is the first step. The other version control operations are for current entries in the system.</p>
Check Out	<p>Verifies if the file is checked out by other users.</p> <p>If the Check Out operation is on a group node, for example, the XLT Configurations node, then there is no behavior change.</p> <p>If Check Out is on a single file, then the version is checked out and the file is directly opened with the corresponding tool.</p> <p>When a file is already checked out by a user, you cannot check out the file. The Check Out menu is disabled. Other users can only open the file in read-only mode.</p> <p>For example, if a user has opened a file in read-only mode, then when a user attempts to check out the file, a check is made to verify if the file that the user has already opened is the latest version. If not, then an error dialog box opens, stating, "The opened file is not the latest version. Close the tool and try again."</p> <p>In this instance, the user must close the opened tool and attempt to check it out again. This guarantees that the file which the user edits is the latest version.</p> <p>If an already opened file is the latest version, then there is no error dialog box.</p>
Replace With	<p>Rolls back the runtime configuration to an earlier version.</p> <p>This lists all versions of the current configuration file, with options to view comments that are associated with each version.</p> <p>To edit a configuration that is based on a previous version, select Replace With to roll back the configuration to a previous version. After the editing is finished, check it in to build a new version.</p> <p>If you only require the runtime configuration to be the previous version, then you can do a Replace With operation.</p> <p>The replaced version is not recorded. This operation is only a modification to the current configuration. The configuration version is not changed.</p>
Discard Change	<p>Discards the current change and rolls back to the previously saved configuration.</p> <p>To give up the current modification, do a Discard Change operation. The current configuration is rolled back to the latest version.</p>

Option	Description
Discard Check Out	<p>Unlocks the file and reverts to the latest checked in version. If the file is checked out by other users, then this is disabled.</p> <p>If the current configuration is checked out by you and you require to release the lock, then you can do this operation.</p>
Show History	<p>Opens the History Versions dialog box. This shows all versions of a selected configuration file, including version number, comments, revision time, and user name.</p> <p>A message is shown on the dialog box with the latest version operation.</p>

Changing the mode

You can change the mode for version-controlled files and non-version controlled files:

- For version-controlled files:

When a user selects **Change to Edit Mode**, the latest version of the file is checked out, and the tool is changed from read-only to edit mode.

A check is made to verify if the file has already been checked out by others. If it has, then a dialog box opens stating "Cannot change to edit mode. Configuration *name* has been checked out by a user".

Another check is made to see if the opened file is the latest version. If it is, then the check out is of the latest version. Otherwise, a dialog box opens stating "Cannot change to edit mode. The opened file is not the latest version. Please reopen the file and try again".

If the file was not checked out by other users and the user's opened file is the latest version, then the file is checked out and the tool is changed to edit mode.
- For files that are not version-controlled:

A check is made to see if the file has been opened in edit mode by other users. If it has, then a dialog box opens to inform the user that the file has already been locked by another user and asks if you require to break the existing lock. If **Yes** is selected, then the mode is changed from read-only to write.

Comparing versions

To compare versions, use **Compare** on the **History Versions** dialog box.

Note: The **Compare** button is enabled when you configure **Compare** with a command line in the **Client Preferences** dialog box.

- On the **Client Preferences > External Command** tab, configure the external compare command by specifying the command in the **External File Compare Command** field. There must be no blank spaces in the installation path of the compare tool.

For example, the external command and path is C:\WinMerge\WinMergeU.exe.

The external file compare command is set to: C:\WinMerge\WinMergeU.exe @F1 @F2

- 2 On the **History Versions** dialog box, enable **Compare** by selecting two versions. If the external compare command is not configured, then **Compare** is not enabled.
- 3 Click **Compare**. The External Compare tool is displayed comparing the specified versions.

Version status

This table shows the version statuses for each configuration node on the Site Manager:

Status	Description
Check in	Parenthesis around the version number indicate the selected version.
Modified Check in	Parenthesis around the version number indicate the selected version. An asterisk indicates the configuration has been modified but not checked in.
Check Out	Angle brackets around the version number indicate the configuration file has been checked out.
Modified Check Out	Angle brackets around the version number indicate the selected version. An asterisk indicates the configuration file has been modified but not checked in. With this status, only the owner can save the file.

Instant version status

When the version of one file is changed by a user, the corresponding node in the Site Manager in all IDEs of those that were opened by several users are immediately updated.

Example:

User1's Site Manager lists "NetConfig (1.18)", where the version is 1.18. This file is not checked out to anyone because at the bottom of the IDE is the status "administrator CHECK_IN NetConfig on version 1.18".

Another user then checks out the NetConfig. The NetConfig node status of User1's client is immediately updated, and now states "yhk CHECK_OUT NetConfig on version 1.18".

There is no requirement to click **Refresh** on the Site Manager to get the latest version status.

The pop-up menu on the node also changes based on the latest version status, so that if User1 right-clicks the NetConfig node to open the pop-up menu, options such as **Check In**, **Check Out**, **Replace With** are not available.

Version Control notifications

When Version Control is configured by any user, all other users receive a notification in the client.

For example, a user checks in NetConfig and the latest version is 1.19. All other users receive in the IDE status bar a notification regarding the operation. This notification contains the user name, operation, file, and version.

When a user double-clicks a message on the status bar, the **Version Control Notification** dialog box opens. This lists all notifications this client has received from the server. These are the operations that took place after the IDE started.

Version Control notification filter

Users can set a Version Control notification filter to notifications. This setting is located on the **Client Preferences > General** tab. The Version Control section of the dialog box displays the current **Notification Filter**. This is used to set what kind of notifications are accepted.

- If no filter is set, then all notifications are accepted.
- If a filter is set, then only notifications for the selected type are accepted.

Clicking **Select** opens the **Notification Filter Selection** dialog box. On this dialog box, you can select the file types and notifications to be accepted by the client.

Note: Notifications for BOX are not implemented.

Examples

This table shows some examples of working with version control.

Examples	Description
Example 1	To see a list of all versions of the Cloverleaf translation file <code>adt_translate</code> , select the file and choose Show History . Cloverleaf displays all versions of the selected file.
Example 2	<p>To make a modification to a Cloverleaf translation file, check out the <code>adt_translate</code> XLT to make modifications. When the file is checked out, and a user selects <code>adt_translate</code> and attempts to check it out, that user is notified that the file is checked out. In this situation, the only options are to open it as read-only or quit.</p> <p>Updates are made to <code>adt_translate</code> and the updates are checked in.</p> <p>Cloverleaf sources the file and applies a version that can be selected later. The file is now available for check-out by other users.</p>
Example 3	<p>To make a modification to a Cloverleaf translation file, a user 1 checks out <code>adt_translate</code>.</p> <p>User 1 does not check back in <code>adt_translate</code> and is out of the office. User 2 needs <code>adt_translate</code> to complete the modifications but the file is locked as checked out.</p> <p>User 3 uses the Server Administration tool to unlock <code>adt_translate</code>. User 2 can now check out <code>adt_translate</code> and complete the changes.</p>

Examples	Description
Example 4	<p>To roll back the Cloverleaf translation file <code>adt_translate</code> to an earlier version, select the file and choose Replace With.</p> <p>Cloverleaf displays all versions of the selected file. You can now select any of the versions to do the replacement.</p>
BOX example 1	<p>User 1 must cut and paste a BOX from the current location to another one in the GUI.</p> <p>If the BOX is already checked out by User 2, then the paste is not performed. An error message opens to inform User 1 that the selected box is checked out by User 2.</p> <p>If the BOX is checked out by User 1 or is not checked out, then the paste is performed. The BOX and its versions are moved to the new location.</p>
BOX example 2	<p>User 1 must copy and paste a BOX in the GUI.</p> <p>The BOX can be pasted to any destination, but the original BOX's version information is not copied to the new BOX. This is regardless of whether the BOX is checked out by User 1 or others.</p>
BOX example 3	<p>User 1 must delete a BOX from the GUI. If the BOX is already checked out by User 2, then the delete is not performed. An error message opens to inform User 1 that the selected BOX is checked out by User 2.</p> <p>If the BOX is checked out by User 1 or is not checked out, then the delete is performed. The BOX and all version information are removed. If User 1 creates a new BOX of the same name as the deleted BOX, then the version information is empty.</p>
BOX example 4	<p>User 1 must rename a BOX from the GUI. If the BOX is already checked out by User 2, then the rename is not performed. An error message opens to inform User 1 that the selected BOX is checked out by User 2.</p> <p>If the BOX is checked out by User 1 or is not checked out, then the BOX rename is performed and all version information is preserved.</p>

Client preferences

Separate `.ini` files are used for users and server administrators. This lets users configure IDE and stand-alone dialog box aesthetics and options. Administrators can configure server parameters and know when to run select administrative GUIs based on the security mode.

These options aid users in dialog box configuration. Some of these options affect an entire site (the client IDE and stand-alone GUIs running from any machine); others affect only clients running from a specific machine.

Select **Options > Client Preferences** to access the **Client Preferences** dialog box. These options affect the client IDE and stand-alone GUIs launched from a specific machine.

Font and document options

This table shows the available options on the **General** tab:

Option	Description
Font Size	<p>This applies to all dialog boxes.</p> <p>The font size is available from 7 to 20. Use the up or down arrows to select the size.</p>
Number of Recent Documents	<p>This indicates the number of files to remember for recent history purposes.</p>
Use different font for master site and root resources	<p>When this is selected (default), all text fields and lists use another font for configuration names acquired from the master site and root resources. Font rules are:</p> <ul style="list-style-type: none">• Master site: black and italic• Root resources: blue and italic• Current site: black and no italic <p>If the check box is not selected, then all text fields and lists always use common font, black and no italic, regardless of where those configurations were acquired.</p>
Automatically connect to the last connected host and site when the client starts	<p>When this is selected (default), the client automatically connects to the last host and site.</p> <p>When this option is cleared, you must manually select the host to connect with.</p>
Open local help document	<p>For users who have a remote IDE, the online help is accessed remotely. Select this to view the help.</p> <p>Note: This option is not necessary when using the portable client.</p>

Advanced Network and JVM options

This table shows the available options on the **Advanced** tab:

Option	Description
JVM Arguments	<p>In this field, you can pass in command-line arguments.</p> <p>Arguments can be passed to the JVM during invocation. The arguments are specific to a JVM. These options vary by operating system. Specify <code>java -h</code> on the command line to see the options.</p> <p>Note: If other items are included on the JVM Arguments line, then they must be separated by spaces.</p> <p>The default maximum heap size is half of the physical memory up to a physical memory size of 192 MB. Otherwise, one-fourth of the physical memory up to a physical memory size of 1 GB.</p> <p>For example, if your machine has 128 MB of physical memory, then the maximum heap size is 64 MB. A greater than or equal to 1 GB of physical memory results in a maximum heap size of 256 MB.</p> <p>If this is blank, then the default maximum heap size is 256MB. If the maximum heap size is not enough for the client, then you can specify an increased maximum heap size using <code>-Xmx</code> (for example, <code>-Xmx512m</code>).</p>
Title	<p>If you require another title on the IDE, then specify the name in the field.</p> <p>Customize the IDE title with these variables:</p> <ul style="list-style-type: none"> • <code>@site</code> shows the site name. • <code>@version</code> shows the IDE version. • <code>@host</code> shows the host name. <p>This is the default value.</p>

XSession options

When the host server is installed on a UNIX machine, organizations sometimes elect to access the host server through an exported display. Access can also be through an XSession package, such as EXCEED, which is installed on Windows clients.

XSession settings differ from one machine to another, and most machines run optimally using the XSession defaults. If, for some reason, your XSession performance is sluggish, then see the documentation for your XServer or contact your system administrator.

You can also try other preset configurations on the **XSession** tab. The best results are obtained by changing the specific XServer settings.

This table shows the available options on the **XSession** tab:

Option	Description
Presets	<p>Local Display is double buffered (BLIT).</p> <p>XSession1 is no buffering (SIMPLE).</p> <p>XSession2 is double buffered (SIMPLE).</p> <p>XSession3 is no double buffering (BACKINGSTORE).</p> <p>Custom is used for other combinations.</p>
Use Double Buffered Components	This makes the repaint smoother. It paints the component into an off-screen buffer and then copies it into the component's on-screen area.
Scroll Pane Mode	<p>BLIT is the fastest mode.</p> <p>SIMPLE uses the method of redrawing the entire contents of the scroll pane each time it is scrolled. This is the slowest.</p> <p>BACKINGSTORE draws view area contents into an off-screen image. This mode may offer advantages over BLIT mode, but requires a large amount of extra RAM.</p>

Note: The defaults are suggested settings.

Remote display IDE

If you require a remote display IDE from a Linux to a Windows machine, then some input fields in the GUI could become uneditable.

To prevent this, you can explicitly set the Toolkit when using Xming. This must be set before starting the virtual machine. To do this, set the Toolkit for an application by Xming or MobaXterm.

Xming has these options:

- Use an environment variable.

```

csh:
setenv AWT_TOOLKIT XToolkit #selects the XToolkit
setenv AWT_TOOLKIT MToolkit #selects the MToolkit
ksh/bash:
export AWT_TOOLKIT=XToolkit
export AWT_TOOLKIT=MToolkit

```

- Use a system property on the command line.

```

java -Dawt.toolkit=sun.awt.X11.XToolkit MyApp
java -Dawt.toolkit=sun.awt.motif.MToolkit MyApp

```

Xlate Config options

If a message contains recursive definitions, then the tree in the Translation Configurator contains nested structures that repeat infinitely.

This table shows the available options on the **Xlate Config** tab:

Option	Description
XML auto-expansion level	<p>This controls how deep into an XML message the Translation Configurator displays when the message definition is initially shown.</p> <p>The dialog box builds the tree up to a certain level and then stops.</p> <p>To manually expand beyond the set limit, click each node in the tree.</p>
JSON auto-expansion level	<p>The JSON format supports auto-expansion levels in the Translation Configurator.</p> <p>The default value is "5". The maximum value is "99".</p> <p>The default maximum number of loaded JSON tree nodes is controlled by the key <code>max_json_tree_nodes_loaded</code>. This is located in the <code>General</code> section of <code>client.ini</code>.</p> <p>On the Translation Configurator, you can select "JSON format" as the Input Format or Output Format. The JSON tree expands according to the selected auto-expansion level.</p> <p>If the loaded number of JSON tree nodes exceeds the maximum value of the default value (5000), then the loading stops and a warning message opens.</p>
Index notation	<p>When this is selected (default), all the fields or composite data structures of HMD formats are addressed by the index notation. For example, <code>0(0).EQU.#1</code>.</p> <p>When the check box is unselected, the addresses for all fields or composite data structures are represented in the original way.</p>
Automatically apply action properties change	<p>When this is selected, Apply is not available on the Translation Configurator Action Properties pane.</p> <p>All modifications are automatically applied. By default, this option is cleared.</p>

Option	Description
Automatically expand operation statement upon entry	<p>This automatically expands all multi-level and nested IF and ITERATE ("+") statements on the Translation Configurator.</p> <p>When this is cleared, buttons are located on the Translation Configurator's New Action toolbar. These are used to expand or collapse all "+" operation statements.</p>
Automatically refresh the target message during xlate debugging	<p>When performing xlate debugging, selecting this option causes the target message to automatically refresh. By default, this is "on".</p> <p>See Xlate Debugger.</p>
Remember the last selected formats	<p>This is convenient when creating a new translation configuration and the last selected format is used. This is saved in <code>client.ini</code> as <code>xlt_remember_last_format</code>.</p> <p>When this option is selected, the last selected input/output record formats are saved in <code>clide.ini</code> with the keys <code>xlt.inbound.format</code> and <code>xlt.outbound.format</code>.</p> <p>For example, <code>xlt.inbound.format=hl7 2.7 ADT_A01</code>. Subsequent translation configurations have the saved values already selected.</p>
Default Timestamp Precision	<p>You can set the default value of the Add Precision property of the DATECOPYOPT action in Translation Configurator with the Default Timestamp Precision selection.</p> <p>When you select On or Off, the Add Precision check box on the DATECOPYOPT action dialog box is checked/unchecked.</p>

External commands

This table shows the available options on the **External Command** tab:

Option	Description
<p>Local Shell Command</p> <p>Remote Shell Command</p>	<p>Use these to customize the command line to specify which program to run when Shell Window is selected under Runtime.</p> <p>The Shell Window provides access to the command line prompt, where all utilities are available. When you click the Shell Window tool, the command line in local mode is the windows command prompt (Xterm in UNIX).</p> <p>In remote mode, it is telnet to host server. This is the default.</p> <p>If you do not configure the Shell Window, then the command line is the default.</p> <p>To customize the command line, specify your preferred command line in Local Shell Command or Remote Shell Command. You can also click the folder button to open a file browser and navigate to the command file for the local machine.</p> <p>For external shell programs selected for local servers, the IDE might or might not successfully pass all environment settings. This includes the current working directory.</p> <p>To configure the command line to access the computer system that runs the host server, specify a @H placeholder in the command line. For example, to configure a command line to telnet the computer system that runs host server, specify <code>telnet @H</code>. The @H placeholder is substituted with the host name of the host server when the command line is run.</p> <p>If an error happens, then when you select Shell Window a message box opens showing exception information.</p> <p>These customized command line values are stored in the General section of <code>client.ini</code>. These values are located at <code>local_shell_window=</code> and <code>remote_shell_window=</code>.</p>
<p>External Editor Command</p>	<p>Select an external editor command by clicking the folder button to navigate to the external editor's folder.</p> <p>External editors are used to create new or open existing Tcl scripts. You can import the modified or generated Tcl script files back to the IDE using the Import Script tool.</p>

Option	Description
External File Compare Command	<p>This is a way to launch a third-party binary to manually compare resource files.</p> <p>Specify the full path of the file compare binary and arguments to the text field. For example:</p> <pre>C:\diff.exe @F1@F2 /i /p</pre> <p>When the command line is run, the @F1 placeholder is substituted with the BOX resource file path. The @F2 placeholder is substituted with the resource file path of the current site.</p> <p>Click the folder button to open a file browser. After selecting a file compare binary, click Open. This closes the browser and populates the field with the selected file path with the two placeholders mentioned above.</p>

NetConfig/NetMonitor options

This table shows the available options on the **NetConfig/NetMonitor** tab:

Option	Description
Automatically update inter-site routing status	Each inter-site thread has a public icon in the upper right corner. For referenced threads, the public icon has a yellow background. For unreferenced threads, the public icon has a gray background.
Enclose thread name in graphics	Inter-site routing auto-updating adds a configuration object listener to the remote server. You can use this check box to turn off inter-site routing auto update. By default, this is unchecked. If a change is made to this setting, then it is not necessary to close and re-open the IDE for the change to take effect. This places the thread name in the protocol graphic in Network Configurator. When this option is cleared, the thread name is placed below the graphic.
Ignore thread bitmap	<p>When this is selected, the threads always show their system-defined protocol images, whether they are configured to use user-defined bitmaps.</p> <p>When this is cleared, the threads show the user-defined protocol images, if any are configured. Otherwise, the threads show the system-defined bitmaps.</p>

Option	Description
Keep original icon size	Inter-site routing auto updating adds a configuration object listener. The thread bitmap is resizable. When this option is selected, the icon is kept the original size.
Use uniform colored background	This pertains to the background of the NetConfig/Net-Monitor tool. By default, the background is a gradient colored background. When the check box is selected, the background is a uniform color.
Route line style	Route lines can be a straight line, or a z-style line. The z-style line has bends and rounded corners.
Enable SMAT file directory out of range warning	The default is selected. When this option is selected, a warning message displays on the Network Configurator's Verifications tab when a thread is directed to a destination that is not <code>HCIR00T</code> . This behavior is the same for outbound message threads in NetConfig.
Automatically load engine output	By default, the process output is hidden. Select this option to view the output.
Show thread bitmap	This setting is saved in the view file and can be loaded again. The Network Configurator ignores this setting, even though the Network Configurator and Network Monitor share the same view file. This shows the selected thread bitmap in the view. By default, this option is unchecked and the Network Monitor view does not show the thread bitmap.
Advanced	<p>You can configure the NetConfig inbound/outbound message file name to populate with a default value by configuring the default value. Click Advanced to open the Configure Default Values dialog box. Select an option from the , or specify one in the field.</p> <p>This shows the selected thread bitmap in the view. By default, thisFor example, with a thread named <code>conn_1</code>, selecting this option and the values <code>@threadname@_in</code> and <code>@threadname@_out</code>, the file names automatically populate with <code>conn_1_in</code> and <code>conn_1_out</code>.</p>

Client SMAT options

This table shows the available options on the Client Preferences **SMAT** tab:

Option	Description
Add all messages to view automatically	When this is selected, all messages in the SMAT file are automatically added to the view after the SMAT file is opened. Otherwise, it opens with an empty view. Adding all messages to the view does not degrade the performance. It only changes the flags of the messages in memory and shows the first message in the view.
Restore last view automatically	When this is selected, the last view definition that is recorded in the <code>ecd</code> file is automatically applied to the view. This can affect performance when loading a SMAT file. The entire message set in the SMAT file is searched using the conditions defined in the applied view definition.
Enable Instant Search	In a regular search, you start the search by clicking Refresh Search . When Enable Instant Search is selected (default), searching is automatic when the criteria or filtering conditions are changed. If performance issues result when using Instant Search, then use the regular search.
Sort search result by Time Saved descending order	This is the default order for all SMAT queries. When this option is selected, the SMAT tool gets by default the newest messages, in descending order.
Select all files when launching SMAT Database Tool from Launch Bar or Tools menu	This lets you configure the SMAT Database tree's default selection. By default, this option is cleared. There is a <code>smat_select_all_db_files</code> property key in the <code>client.ini</code> file that retains the user's setting for this option.

Alert config options

The only option is **Automatically apply alert properties change**.

Click **Apply** on the Alert Configurator's Trigger pane to apply the new action properties to the alert.

By default, this option is disabled. When this option is enabled, **Apply** is not available on the Alert Configurator's Trigger pane. All modifications are automatically applied.

Adding external tools to the IDE

You can add external tools to the IDE using these steps. When these steps are completed, a new `externaltools.xml` file is located under `$HCIR00T/client`.

Note: This feature is only supported from the GUI; otherwise, only the default icon is used for the tools.

Supported icon formats include `bmp`, `jpg`, `jpeg`, `png`, and `gif`.

When an external tool is created, you can also add a script that runs `bat` and `ksh` scripts. With this functionality, you can create a tool that, for example, can start or stop a site.

You can create a `bat` file or create a new external tool. You can also add the `bat` file to the tool, and run the `bat` script in the IDE.

- 1 Launch the IDE and change to the test site, for example, **helloworld**.
- 2 Open the Client Preferences GUI and go to the **External Tools** tab. This tab has these columns: Tool Name, Path, Icon, and Category.
- 3 Click **Add**. The **Add Tool** dialog box is displayed. All fields are required.
- 4 Specify a **Name**. For example, **Notepad**.
- 5 Click **Browse** next to **Path** and navigate to the external tool. For example, `%windir%\system32\notepad.exe`.
- 6 Select the tool's `.exe` file and click **OK**.
- 7 The **Icon** field is optional. If this is left blank, then a default icon is used.
- 8 Select the image file and click **OK**.
- 9 Open the **Category** list and select a category. You can also specify a new category in the field. Click **OK**. The new tool displays on the dialog box.
- 10 Click **OK** and on the **Client Preferences** dialog box.
- 11 You must restart the IDE for the changes to take effect.
- 12 After restarting the IDE, the new category is now on the selected category's menu.

Launch bar tools options

This tab gives you the option of selecting which launch bar tools to hide/show. When the **Visible** check box for a tool is cleared, the tool does not display on the IDE.

For example, to provide a user interface for healthcare domains, you can hide Edifact, VRL, FRL, NCPDP, HPRIM, HRL, and X12. NCPDP is optional, as this is related to pharmacy.

Script editor options

This table shows the available options on the **Script Editor** tab:

Option	Description
Font Size	This is an integer from 7 to 32, which controls the font size of the displayed code within the editor.
Tab Size	This is an integer from 1 to 32, which controls the width of the Tab key displayed within the editor.
Emulate tabs with spaces	When this is selected, and you press the Tab key to insert a tab in the code, the editor auto-converts the Tab key into spaces.
Auto-Activate Auto-Complete	When this is selected (default), the Auto-Complete window is enabled. If this is cleared, then the Auto-Complete window does not pop up unless you press Ctrl+Space .
Open Auto-Complete Window	This provides the shortcut key to manually open the Auto-Complete window. The default is Ctrl+Space .
Fill with Code Template	This triggers the code template function. For example, if you specify "for" and then press Ctrl+Shift+Space , the editor automatically inputs the code template for the "for" loop.
Jump to Matching Bracket	This is useful for bracket matching, especially when the starting and ending bracket extend across many lines. To use this function, position the cursor immediately after a bracket, then press the shortcut keys. This moves the cursor to the corresponding matching bracket. The Script Editor's scroll bar moves to display the line to which the cursor jumped. The default is Ctrl+J .

Right-click menu

This menu provides these functions:

- Undo/Redo/Cut/Copy/Paste/Delete
- Add Proc opens the **Add Proc** dialog box, where you can specify the new procedure and procedure template. New procedures are appended to the end of the current script file.

For JavaScript and Python, the **Proc Type** drop-down includes:

- Tps
- Trxid
- XltCall
- XltObjects
- XltString
- XltStrings

- Toggle Insert text: Opens the Insert Text Assistant panel.
- Select All: Selects all content within the editor.
- Folding: Expands/collapses code folds.
 - Toggle current fold: Places the cursor within a fold. Clicking this menu collapses/expands the fold.
 - Collapse all comments: Collapses all code comments.
 - Collapse all folds: Collapses all code folds.
 - Expand all folds: Expands all folds.

Note: Some functions of right-click menu functions are also on the main **Edit** menu and toolbar.

Site preferences

These options affect the client IDE and stand-alone GUIs launched in a site from any machine. These options are located in the `siteInfo` file on the host server.

Note: Do not edit the `siteInfo` file without direction from Technical Support!

Tcl log message alignment

Log messages contain time stamp information. If these are not required, then you can turn off alignment. To do this, change this line in `siteInfo` (`$HCIR00T\site\siteInfo`):

```
TclLogAlignmentDisabled=0(default)
```

For example, by leaving this line as the default the engine log shows extra alignment:

```
[tcl :err :ERR /0: reader:03/03/2014 08:51:36] this is a message for test stderr
[tcl :out :INFO/0: reader:03/03/2014 08:51:36] this is a message for test stdout
```

Changing `TclLogAlignmentDisabled =1` causes the Tcl output to show without extra alignment:

```
this is a message for test stderr
this is a message for test stdout
```

The other logs keep the extra alignment:

```
[pd :thrd:INFO/0: reader:03/03/2014 08:51:36] [0.0.1] Placing DATA msg on IB post-SMS queue.
[pd :pdttd:INFO/3: reader:03/03/2014 08:51:36] [0.0.1] IB post-SMS message details
[pd :pdttd:INFO/3: reader:--/--/---- --:--:--] msg: 0x030482B0
[pd :pdttd:INFO/3: reader:--/--/---- --:--:--] msgType : DATA
[pd :pdttd:INFO/3: reader:--/--/---- --:--:--] msgClass : ENGINE
```

General options

Note: The database status can be retrieved from the **SMAT** and **Internal/Error Database** tabs.

This table shows the available options on the **General** tab:

Option	Description
Site Color	This alters the site color line at the bottom of the IDE and stand-alone GUIs. Click Site Color to open the Site Color dialog box.
Site Color Bar Size	Alters the site color bar size.
Site Description	<p>Use to specify comments. For example, if you are working on a particular site and must tell others that the site is locked by another person and is unavailable until a certain date.</p> <p>After specifying a site description, an icon displays on the status bar. Clicking the icon switches the text.</p>
Master Site	<p>Select from the menu the master site for the currently open site.</p> <p>For one site, you can select only one master site. You cannot select multiple master sites in the Site Preferences dialog box.</p> <p>When a master site is set in Site Preferences and a master site is also set in Root Preferences, the Site Preferences setting has higher priority and takes effect.</p> <p>When no master site is set in Site Preferences, the master site setting in Root Preferences is used.</p>

Log History options

This table shows the available options on the **Log History** tab:

Option	Description
Enable Log History	<p>This enables Maximum Number of Log Files, Maximum Logs Total Size, and Compress Command.</p> <p>Selecting Unlimited disables the corresponding text field. Click the button below Unlimited to specify a maximum number/total size.</p> <p>For Maximum Logs Total Size specify a positive integer for the KB value.</p>
Automatic Log Cycling	This cycles MonitorD logs when they get to the specified size. Cycling a file causes the current file to close, renames it to <code>old_name.old</code> , and opens a new save file.

Option	Description
SmatHistory Link Path	<p>Select this to save the <code>smatHistory</code> database files based on thread names. When this option is selected, sub-folders that are named as the thread name are created within the <code>smatHistory</code> folder. History <code>smatdb</code> files are saved into these sub-folders.</p> <p>You can subdivide <code>smatdb</code> files within <code>smatHistory</code> based on thread names.</p> <p>See Symbolic links.</p>
SmatHistory saved base on Thread Name	<p>Select this to save the <code>smatHistory</code> database files based on thread names.</p>

NetConfig options

At times, BOX can take a long time to get started. Most of this time is spent in validation, especially when there are many threads that have host names or IP addresses that cannot be reached.

When one or more threads are selected and a new BOX is created, the system could stall while performing DNS validation.

To shorten the validation time, you can opt to omit network validation.

When validation is not performed, the time required in operations such as saving Network Configurator or creating a BOX is decreased.

These options are available on the **NetConfig** tab:

- **Verify whether the host is reachable in thread**
This disables/enables host reachable validation for a thread.
- **Verify whether the thread is reachable in destination thread**
This disables/enables thread reachable validation for a destination thread.

By default, these options are selected.

Note: You must restart the GUI when these options are cleared (validation is not performed).

Site SMAT options

Note: These options are read-only. Changes are made on the **Server Administration > Host Server databases** tab. See [Host Server Databases tab](#).

The Database Options panel is read-only. This lists the current SMAT database encryption settings.

- Saved into:
 - Saved into: File

- Saved into: Database
- Data Encryption:
 - Data Encryption: Disabled
 - Data Encryption: Enabled

These read-only fields are based on the site level. That is, the configuration is implemented on all processes. Processes within a site save the history database files to the link path.

You can also save SMAT history files outside the site folder by creating a link path to a specific folder/disc. See [Loading SMAT history outside the SMAT folder](#).

This table shows the available options on the Site Preferences **SMAT** tab:

Option	Description
Enable SMAT History	These settings affect the SMAT history behavior for each engine process. Because these settings are loaded after the engine is started, an engine bounce/restart is required for these setting to take effect. Daemons do not require to be restarted because SMAT is written by hciengine and has no effect on daemons.

Option	Description
	Delete the oldest SMAT inbound/outbound history file when history file's number for a thread exceeds number files
	Delete the oldest SMAT history file when history file's total size for a process exceeds number KB
	Delete SMAT history file for a process when it is older than number days

Option	Description
	<p>These options are used to maintain the SMAT history files. When Unlimited is selected, the cycled SMAT files are saved. Otherwise, the earlier files are deleted after the limitation of the file number, disk size, or lifetime is reached.</p> <p>If there are more than one file to compress, then each file is compressed individually.</p> <p>If no command is configured, then the cycling files are not archived.</p> <p>By default it is set to NULL and compression is disabled. The <code>gzip</code> and <code>compress</code> commands are supported.</p> <p>Option descriptions are:</p> <ul style="list-style-type: none"> <p>Delete the oldest SMAT inbound/outbound history file when history file's number for a thread exceeds number files</p> <p>This value controls the maximum number of SMAT files within <code>SmatHistory</code>. When the cycled file number exceed this threshold, the newest cycled file is saved and the oldest is deleted. The time stamp is the indicator for deciding which file is the newest and oldest. This option is on the thread-level. Each thread's SMAT file number limitation is the threshold.</p> <p>Delete the oldest SMAT history file when history file's total size for a process exceeds number KB</p> <p>This value is the threshold of the total size of SMAT files within <code>SmatHistory</code>. When the cycled file is backed up to <code>SmatHistory</code>, the engine calculates the size of all files and compares it to this threshold. When the size exceeds the threshold, the oldest file is deleted. The timestamp is also an indicator. This option is for the process-level. The total size contains all threaded SMAT files. It does not indicate the total size of SMAT files in a special thread.</p> <p>Delete SMAT history file for a process when it is older than number days</p> <p>Age indicates the time stamp of the cycled SMAT file. This option continues saving the cycled files as long as the value is set. When a new cycled file moves into <code>SmatHistory</code>, the engine checks all files by timestamp one-by-one. After the file saving time exceeds this threshold, the file is deleted.</p> <p>When all files are set a value, the engine deals with them with a priority of:</p> <ol style="list-style-type: none"> Delete the oldest SMAT inbound/outbound history

Option	Description
	<p>file when history file's number for a thread exceeds number files</p> <p>2 Delete the oldest SMAT history file when history file's total size for a process exceeds number KB</p> <p>3 Delete SMAT history file for a process when it is older than number days</p>
Compress Command	This compresses the backed-up SMAT files to save disk space.
Save SMAT history based on thread name	The default value is unselected (key value = 0). When this is selected, (key value = 1), the <code>smatHistory</code> database files are saved based on the thread names.
SMAT History Location	The default value is null. Select this to save the SMAT history outside of the site folder. This creates a link path, where you can write the actual location in which to save the SMAT history.

Use Log History Setting

Select this for all the common settings to be synchronized. To use this feature:

- 1 Select **Enable Log History** on **Log History** tab.
- 2 Select **Enable SMAT History** on the **SMAT** tab.

Both history options must be selected to enable **Use Log History Setting**.

You can now apply the common settings of the **Log History** tab to SMAT Log History. The **Compress Command** setting cannot be applied if it is disabled. To enable it, you must clear **Save into Database** and select **Enable SMAT History**. After this is enabled, it can be synchronized with the log history settings.

System Management options

This table shows the available options on the **System Management** tab:

Option	Description
Minimum Available Disk Space Minimum Free Memory	<p>For Minimum Available Disk Space, the engine processes and monitor daemon automatically shuts down when the file system on which the Cloverleaf root is installed has less free disk space than the configured parameter.</p> <p>For Minimum Free Memory, the engine processes and monitor daemon automatically shut down when the amount of memory available to the system is less than the configured parameter.</p> <p>For version 19.1 and above, the application checks physical memory only.</p> <p>For versions 6.2 and below, the application checks all memory (physical + virtual/swap).</p>
Enable Ephemeral Port Range	<p>This enables the Port From and Port To fields. The port range must have at least four ports. For example, specifying a range of 10000-10003 allocates ports 10000, 10001, 10002, and 10003.</p> <p>Valid Port From ports are 1025-65532.</p> <p>Valid Port To ports are 1028-65535.</p> <p>The formula for determining the port range is: Minimum port number = (total number of processes)X2 + (total number of protocol threads) + 1.</p>

Ephemeral best practice

Optimizing ephemeral port usage can reduce Support calls and improve performance.

Ephemeral ports are used to have finer grain control over the **Server > Client** connection. The main use of configuring ephemeral ports is for TCP interfaces to access several ports other than a TCP protocol thread.

Ephemeral port optimization includes consideration of these points:

- When all sites use the same range, for example, 29300-42100, `hcimonitor`d issues can result. For example:

```
[cmd cmd :ERR /0:_hcimonitor_d_:08/08/2019 08:55:36] Invalid command length (>= 8192).
Closing connection 0xb9e38b50.
[aler:aler:INFO /0:_hcimonitor_d_:08/08/2019 08:55:36] Removing alerts and waits for connection
0xb9e38b50
[icl tcpip:DEBUG /0:_hcimonitor_d_:08/08/2019 08:55:36] Tcpip shutting down socket 1600
```

Best Practice: Remove explicit ephemeral ranges or select each site to use its own range of 500-1000 ports, based on site size.

- Ephemeral ports give you fine grain control over the Server-Client relationship. By configuring these ports, you TCP interfaces have access to several ports other than a TCP protocol thread.

Best Practice: Do not use TCP ports in Cloverleafs' protocols that are in the operating system's ephemeral range.

- The port range is implementation dependent. Factors to consider include the number of TCP interfaces, multi-threaded TCP/IP interfaces, ICL communications, protocol threads, `hcmontord` connections, and number of processes.
Best Practice: Configure a beginning and ending port range.
- Each operating system uses a unique port range. For example, a range of 32768-61000 forces Cloverleaf to use that range and not use the operating system's defined range. This range is large enough for a given Cloverleaf site.
Although multiple Cloverleaf sites can use the same range, this is not recommended as it circumvents the design and usage of the Cloverleaf ephemeral range. Each site must have its own range that does not interfere or coincide with other Server ports on the Server.
Best Practice: Employ a large range of ports.
- Using a narrower range of a 500-1000 should reserve enough connections for a given Cloverleaf site; however, each site must be tailored by its usage.
Best Practice: Do not have any TCP servers in any Cloverleaf site that are in this range, since an ephemeral port could be using the port.
- To decide which port numbers to use, you must ensure a wide enough range is used to accommodate all of the traffic for the site's interfaces.
Best Practice: Use a range that is not a reserved range for other applications or interfaces, and where there are no Cloverleaf interfaces configured to listen.
- A Cloverleaf site's ephemeral port range is customizable. For customization, you can configure the port range to not use the operating system's predefined range.
Best Practice: Configure the port range to not use the operating system's ephemeral range. This is typically in the range of 32768 to 60999. For example, on RHEL you can use `cat /proc/sys/net/ipv4/ ip_local_port_range`.
- You must determine how large to make the range. For example, a small range of 40000-40004 could cause issues. The range depends on the site's configuration.
Best Practice: The system requires at a minimum: $(\text{Total Number of Processes} \times 2) + (\text{Total Number of Protocol Threads} + 1)$.
- When Implementation determines that an ephemeral port range must be set and the range specified opened in a firewall, or other reasons, then the Implementer decides the range size to use.
Firewalls do not typically restrict outbound traffic, or internal traffic to the operating system traffic.
Best Practice: The system requires at a minimum: $(\text{Total Number of Processes} \times 2) + (\text{Total Number of Protocol Threads} + 1)$.

Error handling

Each check box on this tab corresponds to a message EO sub-module. The default is unchecked.

Selecting a check box elevates warnings that are thrown in the given message sub-module to errors.

These flags are stored as entries in the `siteInfo` file:

- `elevateWarningToErrorForRecParse=NO`

- `elevateWarningToErrorForRecDefLoad=NO`
- `elevateWarningToErrorForRecFormat=NO`
- `elevateWarningToErrorForRecDefQuery=NO`

They display after the current **LogHistoryCompressCommand** entry. The default is "NO", where the engine treats warnings in these categories as warnings. It prints a "WARN" message to the engine log, returns a warning message to Tcl, and then continues processing.

If they are set to "YES", then warnings in these categories are treated as errors. The "ERR" message is printed to the engine log, processing stops, and the message is transferred to the Error DB.

This table shows the available options on the **Error handling** tab:

Xlate Route Details dialog box

In Network Configurator, an **Xlate Route Details** dialog box opens when you click an xlate Type on the **Route Messages** or **Route Replies** tab. You can use this dialog box to indicate whether to error out the transaction. For example, in case there are warnings when parsing the incoming HMD message (**Elevate warnings to errors . . .**). Selecting this elevates warnings from the Message RefParse sub-module to errors on a per-route basis.

When this is selected, it overrides the `RecParse` Setting (**Message Parsing**) in Site Preferences. If cleared, then it inherits the value from the **Site Preferences** dialog box.

The settings are stored as a new route detail entry in the NetConfig file.

The error control settings that are configured in individual translation actions override the site preferences and xlate route details settings.

For example, `elevateWarningToErrorForRecParse=NO`, `ELEVATEWARNINGFORPARSE=1`, `error=skip`, and there is a parse problem during a COPY action. The engine ignores the error because the error setting in the translation action takes precedence.

Alert configuration options

This tab is where you can configure alert-related preferences.

This table shows the available options on the **Alert Configuration** tab:

Option	Description
Global Polling Interval	<p>The global polling interval helps in diagnosis and tuning of the monitor daemon.</p> <p>The global polling interval affects the monitoring updates to the Network Monitor. Extending the polling interval means that the polling interval is the de facto minimum duration. For example, the polling interval is 5 minutes, but an alert is defined with a duration of 30 seconds. The alert does not fire until the next interval, which is 5 minutes.</p> <p>The default value is 5 seconds. The minimum interval is 5 seconds and the maximum interval is 5 minutes. Values greater or lesser than these cause an error.</p>
Email action options	<p>Email options can be predefined though the Site Preferences dialog box. This reduces the requirement of specifying them every time when configuring the email action on the Alerts dialog box.</p> <p>Predefined values are only used when these options do not have specified values in the current alert.</p> <ul style="list-style-type: none"> • SMTP Server is required. Some email servers support SMTP over SSL, port 465. When this happens, emails cannot be delivered, as this is not supported. • Port is the mail port to use. The default is 25. This is not required. • Use TLS to send email: Select when using a secure mailbox. • Login User is required. • Password is not required. • From is required. <p>In the Alert Configurator, separate multiple entries in the To and Cc fields with a comma. For example, someone@domain.com,person@thedomain.com,support@yourdomain.com.</p>
Test	<p>Click this to test whether the currently configured email settings are valid. This opens the Test Email Settings dialog box, where you specify the address and message body of the test email.</p> <p>The Test email Alert supports global variables. The IDE replaces the global variable with the real value before sending the test email.</p>
Only list the information for threads in question in alert message	<p>By default, alert email messages list all thread information in the source. Select this option to only list the threads that meet the criteria.</p>

Option	Description
Write alert event log to Database	<p>Select this to write alert events to an external database (SQLite) instead of the alert log file.</p> <p>A SQLite database is generated under <code>\$HCISITEDIR/exec/hcimondord/alerteventlog.sdb</code>.</p> <p>Every triggered alert is logged in the table of this database.</p> <p>Any SQLite administration tool can access this database file. In Cloverleaf, this database/table is used most often by Global Monitor for end-user visualization on alert logging.</p> <p>This also saves events without a notify alert action.</p>
Default Alert configuration Monitor Daemon uses	<p>Select this to configure the default alert file that Monitor Daemon uses when starting.</p> <p>Only one file can be configured.</p> <p>The default file is <code>default.alert</code>. When this is selected, <code>defaultalertconf=xxx</code> is added to the <code>siteInfo</code> file after saving the configuration.</p>

Database Configurations options

On the **Database Configurations** tab, you can add, update, and delete database connection settings.

The **Database Connection** menu lists all stored database connection configurations. Changing a selection updates all other component values according to the selected connection.

This table shows the available options when a database connection is selected and **Edit** or **New** is clicked.

The remainder of the fields are populated from the **Add New Database Connection** or **Edit Database Connection** dialog box.

When configuration is completed, clicking **OK** stores the connection properties in `$HCISITE\dbconfiguration.ini`.

This table shows the available options on both dialog boxes:

Option	Description
Database Connection	Specify a unique name for the connection. A duplicate name prompts an error message.
Database Connection Timeout	Sets the maximum time in seconds that a driver waits when attempting to connect to a database after the driver has been identified.
Database	Select a database type from the menu.

Option	Description
Database URL	<p>This defines the JDBC URL for the current connection. When a new database connection is added, Database URL must be in the same format as URL Format.</p> <p>Default values are shown in this field. Values in red must be configured.</p>
URL Format	<p>This shows an example of the JDBC connection URL for the selected database. You can refer to this when specifying the Database URL.</p> <p>The default port for Oracle is 1521 and for SQL Server is 1433. These are used when specifying the Database URL.</p>
User Name/Password	<p>These defines the valid user name/password with which to connect to the database.</p>

Option	Description
Schema	<p data-bbox="818 302 1409 432">This sets the database schema for the current connection. This setting indicates that all the SQL Statements running in the current connection should be available under this schema.</p> <p data-bbox="818 449 1409 548">Click List to open the Schema selection dialog box, from which you select an available database schema for the current connection.</p> <p data-bbox="818 564 1386 663">This field is optional. For the oracle database, if schema is not set, the IDE uses User Name as the default schema.</p> <p data-bbox="818 680 1409 842">When configuring a Database Inbound/Outbound protocol in Network Configurator, the schema must be listed for the table/view name in the SQL statement. This is only if the selected schema does not belong to the selected user.</p> <p data-bbox="818 858 1409 1020">If the listed schema in <code>dbconnection.ini</code> does not match the default schema in Schema, then this schema name must be used as the table name prefix in the SQL statement. This is the statement that is used in the Database Inbound/Outbound protocol.</p> <p data-bbox="818 1037 1409 1167">For example, if the default schema that is shown on the Database Configurations tab for user1 is schema2, then the <code>dbconnection.ini</code> configuration is:</p> <pre data-bbox="833 1215 1013 1260">schema=schema1 userid=user1</pre> <p data-bbox="818 1304 1409 1402">On the SQL statement of the Database Inbound/Outbound protocol, <code>schema1</code> must be placed as the prefix to <code>tablename</code>:</p> <pre data-bbox="833 1446 1227 1467">SELECT * FROM schema1.tablename</pre> <p data-bbox="818 1512 1409 1570">Note: <code>tablename</code> is under the schema <code>schema1</code> in the external database.</p> <p data-bbox="818 1587 1409 1646">If <code>schema1</code> is not identified in the SQL statement, then an invalid object '<code>tablename</code>' error results.</p>
SQL Statement	<p data-bbox="818 1682 1409 1743">Specify SQL statements in this field. This field is optional. The default value is empty.</p> <p data-bbox="818 1759 1409 1890">You can use multiple lines. These values are saved in the <code>\$HCISITE/dbconfiguration.ini</code> file. Database Connection must have a selection before this can be saved.</p>

Option	Description
Large Object Type	<p>This property (<code>largeobjecttype</code>) identifies the implementation details of BLOB/CLOB in different databases.</p> <p>Options are:</p> <ul style="list-style-type: none"> oracle sqlserver sqlite postgresql <p>The default value for uncertified databases is <code>oracle</code>.</p> <p>This option is only useful if it is another database, not Oracle, SQLServer, SQLite, or PostgreSQL.</p> <p>If the database is one of these, then the option is not available, and is not required. Other databases are configurable.</p>

Adding, editing, and deleting a database driver

Clicking **Configure Database Drivers** on the **Database Configurations** tab opens the **Configure Database Drivers** dialog box.

From this dialog box, you can insert, delete, and update a database configuration.

The **Database** menu lists all stored database configurations. By default, the selected database is the same as the one shown on the **Database Configurations** tab. Changing a selection updates all other component values according to the selected database.

1 On the **Configure Database Drivers** dialog box, click **New** or **Edit** to add or edit a database driver.

2 Specify the parameters on the resulting **Add New/Edit Database Driver** dialog box.

Click **Open** next to **Driver File Path** to open a file browser where you can select the relevant JDBC driver jar file. This sets the jar file's full path in **Driver File Path**. This browser only supports jar file selection and permits browsing under `$HCIRoot`. The default location is `$HCIRoot\clgui\lib`.

For the predefined SQL Server and Oracle databases, the default value of **Driver File Path** must be available after installation. The default location for driver jar files is `$HCIRoot\clgui\lib\jdbc_jar_file`.

Click **OK** to add/update the database connection configuration.

3 All modified database configurations are saved when **OK** is clicked on the **Database Configurations** tab. This action also updates the **Database** menu on the main tab. Database driver configurations are stored at `$HCIRoot\server\database.ini`.

By default, it predefines these configuration entries:

```
[sqlite]
driver_class=org.sqlite.JDBC
driver_file=<HCIRoot>\clgui\lib\sqlite-jdbc.jar
```

```
url_format=jdbc:sqlite:<database file name>
...
[SQL Server]
driver_class=com.microsoft.sqlserver.jdbc.SQLServerDriver
driver_file= <HCIR00T>\clgui\lib\ sqljdbc4.jar
url_format= jdbc:sqlserver://<server>:<port1433>;DatabaseName=<database>
...
[Oracle]
driver_class=oracle.jdbc.OracleDriver
driver_file= <HCIR00T>\clgui\lib\ojdbc7.jar
url_format=jdbc:oracle:thin:@//<server>:<port1521>/<sid>
```

- 4 After creating the database, if you must change any parameter, click **Edit** on the **Database Configurations** tab. The **Edit Database Driver** dialog box opens. In this dialog box, you can update the configuration of the selected driver.
- 5 To delete the database, click **Delete** on the **Database Driver Configurator** dialog box. The IDE first checks whether any database connection is referring to the selected driver. If the driver is never referenced, then a confirmation message is displayed.

Internal/Error database options

Note: These options are read-only. Changes are made on the **Server Administration > Host Server tab > Databases tab**. See [Host Server Databases tab](#).

Option	Description
Internal Databases Options: Data Encryption	<p>This option is read-only.</p> <p>Options are:</p> <ul style="list-style-type: none"> • Data Encryption: Disabled • Data Encryption: Enabled <p>The internal database stores messages in an encrypted form.</p> <p>When this is disabled, the internal database stores messages in plain text. Otherwise, it is password protected. In this case, <code>internaldbkey</code> is used for the internal database.</p>
Error Database Options: Data Encryption	<p>This option is read-only.</p> <p>Options are:</p> <ul style="list-style-type: none"> • Data Encryption: Disabled • Data Encryption: Enabled <p>The error database stores messages in an encrypted form.</p> <p>When this is disabled, the error database stores messages in plain text. Otherwise, it is password protected. In this case, <code>errordbkey</code> is used for the error database.</p>

All database encryption settings are stored in the `HCISITEDIR\siteSecurityInfo` encrypted file.

External database options

This tab is used to configure an external database which stores the archived message and message tracing data. On this tab, you specify the Database Connection, where you select a JDBC connection for the external database. The menu lists all available JDBC connections that are configured through the **Database Configuration** tab.

With message archiving, you can save message data and metadata to an external SQL database such as Oracle.

For any thread, both inbound and outbound messages can be archived. For a configured archive, data is first saved as a temporary message cache. The data is then written using JDBC to a table in the configured external database. The Alerts framework in Monitor Daemon determines when the processes to write messages to the external database are run. To aid in configuration of these alerts, there is a tool that generates these alerts configured using default options.

When configuring an external database, there are certain limitations that must be observed:

- An empty (NULL) message fails to be inserted into an Oracle database table when the Message Data column is defined as "NOT NULL". If you define the column as NULL, then the empty message can be inserted into the Oracle database. The SQL database does not have this limitation.
- Table names are case-sensitive.
- Cloverleaf supports letters, digits, and underscores in table names and column names.

This table shows the available options on the **External Database** tab:

Option	Description
Database Connection	Lists all database connections for the master/current site.
Automatic Log Cycling Threshold	Automatically cycles message logs when they attain the specified size. Cycling a save file causes the current file to close, renames it to <code>old_name.old</code> , and opens a new save file. The save files are located in %HCISITEDIR%\exec\hcimsgarchive.
Unlimited	Select this or specify the maximum number of kilobytes the log file can attain before cycling in the field.
Enable Message Tracing	This enables/disables the message tracing feature. You can specify any number of days.
Table Name	<p>This is a site level configuration. The table name is used, because all tracing data in one site is stored into one table in the external database.</p> <p>For better performance, the system engine records the tracing data to an embedded database before archiving them to the external database.</p>

Option	Description
Message Count Threshold Timeout in Seconds	<p>These are used to specify when the archiving action is triggered. One of these options must be specified. Otherwise, an error message results.</p> <p>Message Count Threshold specifies the maximum number of tracing records stored in the database before being written out to the external database. The default is 100.</p> <p>Timeout in Seconds specifies how often the archiving action is performed. The default is 600 seconds.</p>
Maximum Age of Tracing Records	<p>This specifies the expiration date of the tracing records. This keeps the external database from growing indefinitely.</p> <p>When Unlimited is selected, the tracing records never expire.</p> <p>When Days is selected, the tracing records are purged when the specified maximum number of days is obtained. By default, the maximum age of tracing records is 10 days.</p>

Aliases options

The field segment database and the aliases database on the site level stores user-customized format definitions and aliases.

This tab lists all existing aliases for the current site.

Available options are **Add Alias Item**, **Edit Alias Item**, and **Remove Alias Item**.

You can add/edit/remove a site's field segment definition through the existing HL7/HPRIM tool.

For HL7 2.7 and HPRIM variants, changes on segments and fields are synced to the field segment database in the site.

An alias is created to associate the `segment.field` number to a common name. For example:

- PID.5.1 = Lastname
- PID.3.1 = MRN
- ORC.2.1 = Accession Number

A list of standard aliases is provided. You can also create and save custom aliases on this tab.

Statistic database options

The **Server Administration > Databases** tab is for enabling/disabling engine encryption. Selecting a site enables the encryption options for the internal, error, and SMAT databases for that site. In earlier versions, these settings were on the Site Init tool.

Configuration options are stored at the site level at `$HCISITEDIR/siteSecurityInfo`.

On the **IDE > Options > Site Preferences > Internal/Error Database** tab is a read-only indication of whether the internal and error databases are encrypted.

For the SMAT database, the **Site Preferences > SMAT** tab has a read-only Database Options panel. This displays the state of SMAT data encryption and where it is saved.

Note: The Statistic database supports a cycle similar to SMAT.

Statistic Database tab

This table shows the available options on the **Site Preferences > Statistic Database** tab.

Note: When the Site Documentation is generated, the Statistic History settings display when **Store runtime statistics into database** and **Enable statistic database history** are both checked.

Option	Description
Enable statistic database history	<p>Enables the settings to cycle the old statistic database file. Selecting this option enables:</p> <ul style="list-style-type: none"> • Delete the oldest statistic database history file when history files' number exceeds xxfiles • Delete the oldest statistic database history file when history files' total size exceeds xx KB • Delete statistic database history file when it is older than xx days • Statistic database History Location <p>This is saved to the <code>siteInfo</code> key <code>monitord4stats2d bhistoryenabled</code>.</p>
Store runtime statistics into database	<p>When Enable statistic database history and Store runtime statistics into database are selected, the statistic history related options can be modified.</p>
Statistic Store Interval	<p>Configure an interval for update frequency, or use the default. The default interval is 15 minutes.</p>
Statistic Database Automatic Cycling Threshold	<p>Configure a threshold, or use the default. (40MB)</p>
Statistic Schema List in Database	<p>Select which statistics to store.</p>

Option	Description
Delete the oldest statistic database history file when history files' number exceeds xxfiles	<p>This is the maximum number of backed up statistic database files. This deletes the oldest statistic database history file when the number of history files exceeds the specified number. SMAT cycling is used to back up the files.</p> <p>This is saved to the <code>siteInfo</code> key <code>monitord4stats2dbhistorymaxfiles</code>.</p>
Delete the oldest statistic database history file when history files' total size exceeds xx KB	<p>This is the maximum size in KB of the total backed up statistic database files. This deletes the oldest statistic database history file when the history file's total size exceed the specified number.</p> <p>This is saved to the <code>siteInfo</code> key <code>monitord4stats2dbhistorymaxdiskuse</code>.</p>
Delete statistic database history file when it is older than xx days	<p>The maximum age in days to keep history statistic database files. This deletes the oldest statistic database history file when a history file's age exceeds the specified number.</p> <p>This is saved to the <code>siteInfo</code> key <code>monitord4stats2dbhistorymaxage</code>.</p>
Statistic database History Location	<p>This saves the statistic database history outside of the site folder. This creates a link path, where you can write the actual location in which to save the SMAT history.</p> <p>The default value is null.</p> <p>When the value in the text field exceeds <i>number</i>, this is saved to the <code>siteInfo</code> key <code>monitord4stats2dbhistorylinkpath</code>.</p>

siteInfo file

The `siteInfo` file has a section that is for statistics that are exported to the database.

```
# Monitord statistics information exporting to external DB configuration:
# monitord4stats2db           - exporting engine statistics information to db.
#                               0 - disabled by default.
#                               1 - enabled.
# monitord4stats2dbinterval   - exporting frequency in seconds(15 minutes by default, 15 secs
#                               as min one).
# monitord4stats2dbschema     - stats table list for exporting.
# monitord4stats2dbsizecycle   - stats db file automatic cycling threshold(in KB),
#                               0 - disable auto cycling
monitord4stats2db=0
monitord4stats2dbinterval=900
monitord4stats2dbschema=host,site,processes,threads,interthreads
monitord4stats2dbsizecycle=40960
```

StatsDB is a sqlite database instance file, located in the `monitord` directory under the site. For example: `$ HCISITEDIR/exec/hcimonitord/mdstatsdb.sdb`.

StatsDB cycling is not on every restart, but is an option. The cycled StatsDB file is named with the timestamp `YYYY-MM-DD_HH-mm-SS`. For example `$HCISITEDIR/exec/hcimonitord/statsdb.sdb.YYYY-MM-DD_HH-mm-SS`.

The `cyclestatsdb` command is for manually cycling StatsDB. For example: `D:\ >hcicmd -t d -c "cyclestatsdb"`.

Response is one of:

- `cyclestatsdb {Stats DB cycled}`
- `cyclestatsdb {Statsdb files failed}`
- `cyclestatsdb {Statsdb files failed}atsdb {Statsdb files is not enabled/supported}`

Example

User 1 has defined a new lab interface and deployed it into a new site (lab2). User 1 also needs to enable encryption on SMAT Database. To do this:

- 1 User 1 logs into the **Server Administration** tool and selects the **Databases** tab.
- 2 From **Site**, User 1 selects the **lab2** from the list.
- 3 User 1 then selects **Enable** for the Internal, Error, and SMAT databases for the lab2 selected site.
- 4 User 1 also sets a custom password for each object.

Statistic database tables

This table shows the site stats database table schema:

Table Column	Type	Description
Host	Text	Host name where Cloverleaf is installed
Root	Text	HCIRoot
Site	Text	Cloverleaf site name

This table shows the processes stats database table schema:

Table Column	Type	Description
ProcessName	Text	Process name
DateTime	Integer	Sampling time
State	Text	Proc status
Threads	Text	Threads in process

This table shows the threads stats database table schema:

Table Column	Type	Description
ThreadName	Text	Thread name
DateTime	Integer	Sampling time
State	Integer	Thread state
LastExtractTime	Integer	Last MSI extract time
LastUpdateTime	Integer	Last MSI update time
StartTime	Integer	Thread start time
StopTime	Integer	Thread stop time
ProtoStatus	Integer	Protocol status
ProtoFlags	Integer	Protocol flag set
ProtoLastRead	Integer	Last protocol read time
ProtoLastWrite	Integer	Last protocol write time
ProtoLastError	Integer	Latest protocol error
ProtoErrorMsg	Text	Latest protocol err info
XlateCnt	Integer	Translation count
FwdCnt	Integer	Forward count
ErrorCnt	Integer	Error count
MsgIn	Integer	Number of messages in
MsgOut	Integer	Number of messages out
BytesIn	Integer	Number of bytes in
BytesOut	Integer	Number of bytes out
IBLatency	Real	IB latency
OBLatency	Real	OB latency
TotalLatency	Real	Total latency
IBPreSMSQD	Integer	IB Pre queue depth
IBPostSMSQDh	Integer	OB Post queue dept
OBPreSMSQD	Integer	OB pre queue depth
OBDataQD	Integer	OB data queue depth
OBReplyQD	Integer	OB reply queue depth
PreXlateQD	Integer	PreXlate queue depth

This table shows the interthreads stats database table schema:

Table Column	Type	Description
ThreadX	Text	Sender thread
ThreadY	Text	Receiver thread
DateTime	Integer	Sampling time
NSent	Integer	Sent counter
NRcvd	Integer	Receiver counter
PostXltQD	Integer	Post xlate queue depth
XlateTime	Real	Cost in xlation
TimeOnQ	Real	Time on queue
TotalLatency	Real	Total latency

This table shows the host stats database table schema:

Table Column	Type	Description
DateTime	Integer	Sampling time
TotalDiskSize	Real	Total disk size
FreeDiskSize	Real	Available disk size
TotalMemSize	Real	Total memory
FreeMemSize	Real	Free memory
TotalVMSize	Real	Total VM
FreeVMSize	Real	Free VM

Scheduler

Access the Scheduler at **Admin Tools > Server Configuration > Scheduler Log**.

Functions that can be scheduled are:

-

In the **Scheduler**, you can select a script and schedule the date/time that the script runs.

- Scheduled scripts are displayed in a list.
- List entries can be added, removed, or edited.
- Clicking **Add** opens a file dialog box where scripts are selected.
- After a script is selected, you can then select the date and time to run the script.
- You can change command options and the date/time for any existing event.

- The **Command** drop-down list is sorted.
- You can set an existing event as "Active" or "NOT".

Basic and Advanced Scheduling are supported.

- Basic Scheduling runs every "N" seconds.
- Advanced Scheduling uses syntax similar to `cron` to specify the schedule.

An ACL applies to configuration changes if Advanced Security is enabled.

This is accessible through CLAPI.

Using the Scheduler in BOX Manager

When you create a BOX in the BOX Manager, on the Modify Resources wizard page, you can click **Add Libraries** to import `server\hcscheduler.ini`.

This merges the `root\libraries\server\hcscheduler.ini` node in **Merge Box Resource into Current Site** when deploying a BOX.

Adding a new event

To add a new event:

- 1 Click the **Add** button to create a new empty event record. This creates a new event line.
- 2 Select the new event by clicking the check box. This makes the event active, so that the event runs. When the box is cleared, the event is inactive, and does not run.
- 3 Specify the event name.
This name cannot be a duplicate of another event.
The name is case insensitive.
To remove an event, select the event from the current event list and click **Remove**.
- 4 Click the **Command** drop-down list for that event and select a command. The commands are sorted.
The Command selection is editable, for specifying command options. For example, "-A".
Validation for inline editing happens on cell change.
- 5 Set the schedule for the event. To do this:
 - a Click the schedule setting button. This opens the **Edit Schedule** dialog box.
 - b In this dialog box, select **Basic** or **Advanced** mode.
 - c After configuring the schedule, click **OK**. The schedule now displays in the **Schedule** field.
To reconfigure an event, double-click any cell of an existing event. After this, you can make any updates.
- 6 Click **OK** to save all updates/changes and exit.
Click **Cancel** to discard changes and exit.

Scheduler modes

Scheduler modes are:

- Basic

In Basic mode, you can schedule an event for Every "n" Seconds/Minutes/Hours/Days.

- Advanced

In Advanced mode, you can schedule an event for every:

- Second
- Minute
- Hour
- Day of Month
- Month
- Day of Week

Each field is a single value and can be combined with other values. To do this, use a comma as the delimiter. For example, in the **Hour** field, you could specify "2,5,18". This indicates the command will run at 2:00 AM, 5:00 AM, and 6:00 PM.

To run the command on Monday and Friday, for **Day of Week** specify "1,5".

The complete setting displays in the non-editable field under the **Advanced** button.

Configuring Windows authentication through the SQL server

To configure a Windows authentication log-in user when SQL server and CIS are installed on several machines:

- On the host server machine, change the log-in account of “Infor Cloverleaf(R) Integration Services” windows service to be a domain user.
- On the host server machine, ensure the domain user is added into the same groups as those of local user hciuser.
- On the SQL server database, create a new Windows authentication log-in user for the domain user.
- On the specified Database of SQL Server, add a database user for the log-in user.

To configure a Windows authentication log-in user when SQL Server and CIS are installed on the same machine:

- On the SQL server database, create a new Windows authentication log-in user for local user hciuser.
- On the specified database of SQL server, add a database user for the log-in user.

To configure a database connection through the CIS client GUI, the database URL must be in this format: jdbc:sqlserver://server:port1433;DatabaseName=database;integratedSecurity=true.

During configuration, the user name can be any string, and the password can be empty.

Connecting to a MySQL database

- 1 Install MySQL. In this example, MySQL Connector/J 5.1.23 is used.
- 2 Add a new database driver.
 - a Open the **Site Preferences** dialog box and go to the **Database Configurations** tab.
 - b Click **Configure Database Drivers** and then **Add** to open the **Add New Database Driver** dialog box.
- 3 In the **Add New Database Driver** dialog box, specify:

Driver Class: `com.mysql.jdbc.Driver`

URL Format: `jdbc:mysql://server:port3306/database?profileSQL=true`

Driver File Path: `C:/Program Files/MySQL/MySQL Connector J/mysql-connector-java-5.1.23-bin.jar`

Note: **Driver File Path** is the location where you placed the JDBC driver file.
- 4 Create a new database connection.
 - a Open the **Site Preferences** dialog box and go to the **Database Configurations** tab.
 - b Click **New** to open the **Add New Database Connection** dialog box.
 - c Specify:
 - **mysql_sakira** Connection Name:
 - Database: `mysql`
 - ConnectionDatabase URL: `jdbc:mysql://hostname:3306/sakila?profileSQL=true`
 - User Name: `root`
 - Password: This is the root password.
 - Schema: `sakira`

You must manually specify `sakira` or the name of the database to use.
- 5 Update the **Database Schema Configurator** GUI.
 - a Open the **Database Schema Configurator** and open `mysql_sakira`.
 - b Specify:
 - Database Connection: `mysql_sakila`
 - Click **Import** and select one, some, or all of the tables.
 - Save the configuration.
- 6 Configure a database: Inbound protocol
 - a Open the **Network Configurator** and configure a database -inbound thread protocol.
 - b Specify:
 - Database Connection: `mysql_sakila`
 - Table Schema: `actor`, or one of the imported tables.
 - Read Action: SQL Statement: `SELECT * FROM sakila.actor;`
 - Scope: `Each row as a record`
 - Max Row per Read: `1000`
 - Scheduling: Scan Interval: `30`

Root preferences

A master site is a site that can share its configurations with all other sites under the same host. You can use those shared configurations to specify settings in these sites in the same manner as their local configurations. With this concept, you can maintain and share common configurations and reduces the requirement to duplicate them between sites.

From the IDE you can:

- Set any site as the master site. Every root can have only one master site.
- Maintain and manage the master site as usual. You cannot open and edit the master site's configurations in the current site. To do that, you must switch to the master site.
- Select the master site's configurations in the dialog box selection lists.

Except for NetConfig, all other configurations in the master site are selectable in each site.

When the current site's configuration has the same name as the one in the master site, the dialog box selection list only shows a single instance. The one in the current site takes precedence.

Master site configuration objects are only editable from within the master site, but can be used for runtime for any site. Local sites can reference the master site's configuration files, but cannot open or modify the configuration. To modify the master site definition, you must switch to the master site.

You cannot see the master site files in a local site's Site Manager or any open dialog box where you can open the configuration files. In places where you can reference a configuration file, you can see the master site's item in the list.

For example, a masterHL7 variant is defined in the master site. When you create a new `xlate` file in the local site, you can select masterHL7 as an inbound or outbound format. You cannot see the masterHL7 in the local site's Site Manager HL7 variant list.

Master site

The master site is a central repository that other sites can reference for shared configuration information. Changes to interface objects are dynamically applied to the running Cloverleaf process. This is accomplished by reloading the cache on all running processes.

To support multiple environments on the same machine, you can reference multiple master sites for storage of configuration information. For example, test, dev, and QA environments.

Master site settings for individual sites are migrated forward during site promotion. Root master site settings are not migrated.

Master Site configuration is available through CLAPI.

In the IDE, the master site's configurations are in italics.

If a master site configuration has a name conflict with the current site, then the IDE only shows one entry for the named resource. The one in the current site is the effective one.

The master site feature includes:

- Root-level master site configuration:

When defined in the **Root Preferences** dialog box, the root-level master site applies to all sites by default.

- Site-level master site configuration:

The site-level master site configuration can override the defined root-level master site.

- Master site configuration purge/reload.
- Master sites are only one level deep. There is no nesting.

The master site contain configuration objects that are shared between sites. Shared objects include:

- Alerts
- Engine output
- Message format definitions
- Tcl procs
- Table configurations
- Xlate configurations
- SMAT database/error database auto-complete suggestions

In the Cloverleaf GUI, from **Options > Root Preferences**, you can select which master site all sites reference by default.

From **Options > Site Preferences**, you can select which master site the current site references for shared configuration data. This overrides the **Root Preferences** default.

Information on multiple master sites is also available. See:

- [General options](#)
- [Master site configuration](#)

Reloading objects

The **Reload Objects** function is accessed from the IDE's **File** menu and toolbar. Clicking this opens the **Reload Objects** dialog box, which automatically reloads all configurations at runtime, on all sites in `HCIRoot`.

This includes TCL procs in the TCL interpreter, global variables, translation files, and message format definitions.

The text area displays the command result.

- If an error happens starting the command, then a message displays the error information.
- If the command was started, but an error was found during running the command, then the error information and other normal output is displayed.

Command line usage is also available. See [hcireloadobjects](#).

Master site configuration

Root-level master site:

- This is selected on the **Root Preferences** dialog box.
- No site is selected by default.
- This selection is stored in the root configuration file at `$HCIRoot/rootinfo`.

Site-level master site:

- This is selected on the **General** tab of the **Site Preferences** dialog box.
- The list of sites excludes the current site and includes a choice to not select any site.
- No site is selected by default.
- This selection is stored in the site configuration file at tab of the `$HCISITEDIR/siteinfo`.

The name (Master) is displayed together with the master site name in the Site Manager and **Server and Environment selection** dialog box. This lists all sites on initial log-in and when selecting **Server > Change**.

Master is also shown in these contexts:

- Site name that is shown on the Window title bar. This also shows the selected master site.
- Status bar site name.
- Site folder that is shown on the Site Manager.

Multiple master sites

Multiple master sites can be in one root.

Each site can select their own master site. One site can select only one master site.

A one-level master site is supported. For example, SiteA selects SiteB as master. SiteB selects SiteC as master. Now, only the configuration of SiteB is available for SiteA. The configuration of SiteC is not available for SiteA.

Note: One-level master support indicates one-level master visibility support. That is, only a one-level master configuration is visible or selectable for the current site.

When a master site is configured, the master site name on the **Please select a Configuration Server and Environment (Root/Site)** dialog box has a "(Master)" suffix.

When a master site is opened, the site name at the bottom of the GUI also lists the site name with the "(Master)" suffix.

The "(Master)" suffix is also displayed after a master site name on the Site Manager.

Master site use cases

Multiple master sites

To separate a common integration object into two separate master sites, mTEST and mQA:

- 1 Create a new site "EPIC_ADT_QA".
- 2 Open the IDE and select **Options > Site Preferences**.
- 3 Select "mQA" as the master site for the "EPIC_ADT_QA" site and then click **OK**. The interface objects in the "mQA" site are now available to the site "EPIC_ADT_QA".

Root master site

- 1 Create several message definitions and several Tcl procs in site "common1". These can be used in the other Cloverleaf sites.
- 2 Open the IDE and select **Options > Root Preferences**.

- 3 Select "common1" as the root master site. The interface objects in "common1" are now available to all sites.

Setting a site as master

Any site can be set as the master site from the IDE client.

The IDE client initializes the site list with the master site setting in `rootInfo` as the current selection. If the setting is NULL, or is empty, then an empty value is selected. After the new selection is applied, the IDE client prompts a message. This states the new master site setting is not effective until the open GUIs are restarted.

This message is only prompted when the change is within the current IDE client instance.

Other IDE instances can be connected to the same host. They check if the master site is changed when they retrieve the configurations from the master site.

These include Tcl procs list, Xlate list, variant list, and so on. When this happens, a message prompts you to re-open all tools for this to take effect.

The master site setting is stored in the `%HCIROOT%/rootInfo` file. In this file, an entry is appended under `platform`. For example:

```
platform=0012
mastersite=master_site_name
```

This setting defaults to NULL. After you select one site as the master, the entry value is updated to the corresponding site name.

The restarting delay period is the time waiting period between stop and start when restarting MonitorD, Lock Manager, a process, or a thread. To customize this delay time, in the **Root Preferences** dialog box, specify a time in seconds in **Restarting Delay**.

Engine output configurations

The engine output configuration list consists of engine output defined from the current site, master site, root, and master.

Resources are color-coded. All resources from the root level are in blue italic. For example:

- "enable_**" resources are root-level and are shown in blue italic.
- "sane_eo" is an engine output defined in the current site.
- "site_eo" is an engine output defined in the master site and is shown in black italic.

Server interface

These command-line tools are affected by the master site:

Command-line tool	Description
hcimonitor	When you specify the <code>-cl</code> option to use the alert configuration files, then the master site is searched if the specified configuration files are not found at the site level.
hcuparsetest	When you specify the <code>-F</code> , <code>-v</code> , and <code>-v</code> options to indicate the format, version, and variant, then the master site is searched when the specified configuration files are not found at the site level.
hciedifacttest	
hcihl7test	
hcincdpctest	
hcincdpcripttest	
hcincdpfabtest	
hcihprimtest	
hcixl2test	When you specify the FRL name to use, then the master site is searched if the specified configuration file is not found at the site level.
hcifrltest	
hcivrltest	
hcihrltest	When you specify the HRL name to use, then the master site is searched if the specified configuration file is not found at the site level.
hcixmltest	When you specify the <code>-p</code> option and the <code>ocm</code> to use, then the master site is searched if the specified configuration file is not found at the site level.
hcixlttest	When you specify the XLT name to use, then the master site is searched if the specified configuration file is not found at the site level.

Command-line tool	Description
hcieols.pl	<p>This command lists all current EO alias configuration files in the order of master, root, and site:</p> <pre>C:\cloverleaf_dev\cis6.1\integrator >hcieols disable_all enable_all enable_all_info enable_all_info_1 enable_all_info_2 enable_all_info_3 disable_all enable_all</pre> <p>The same file names are printed multiple times if they are defined in multiple levels.</p> <p>To avoid ambiguities, the level header is printed before printing the EO configuration files.</p> <p>The command to include EO alias is modified from the master site.</p> <p>The <code>-n</code> option shows only the Master Site alias.</p> <p>If the Master Site is not set, then the level header is not printed. On the other hand, if the Master Site is set but contains no EO alias, the level header is printed but with no subsequent entries. For example:</p>

```
[Master C:\cloverleaf\cis6.1\integrator\eoalias\master]
disable_all
enable_all
enable_all_info
enable_all_info_1
enable_all_info_2
enable_all_info_3
[Master Site C:\cloverleaf\cis6.1\integrator\Master
Site\eoalias]
[Site C:\cloverleaf\cis6.1\integrator\Site\eoalias]
disable_all
enable_all
```

Command-line tool	Description
hcifrlls.p hcialertls.pl hcihrlls.pl hcixltls.pl hcitblls.pl hcivrlls.pl	<p>These commands are modified to show the configurations in the master site level:</p> <pre>C:\cloverleaf_dev\cis6.1\integrator\eoalias>hcixltls [Master Site C:\cloverleaf_dev\cis6.1\integrator\mas ter\xlate] Cerner ORM_001_CBORD_Diet ORM_001.win.xlt dt_error.xlt dt_error_xml.xlt mytest.xlt test.xlt test1.xlt [Site C:\cloverleaf_dev\cis6.1\integrator\Site\xlate] test1.xlt test2.xlt</pre> <p>Also, options <code>-s</code>, <code>-n</code>, and <code>-a</code> show configuration files in site only, master only, and both levels, respectively. The default shows both levels if no options are given.</p> <p>Similar to the behavior of <code>hcieols</code>, if the Master Site is not set, then the level header is not printed; if the Master Site is set, yet it contains no EO alias, then the level header is printed but with no subsequent entries.</p>
setroot	<p>These environment variables are added when <code>setroot</code> is called:</p> <pre>HCIMASTER="MasterSiteName" HCIMASTERDIR="MasterSiteDir"</pre> <p>In addition, these environment variables are modified to include the master site directory after the site directory:</p> <pre>CLASSPATH=C:\cloverleaf_dev\cis6.1\integrator\site\java_uccs; \$HCIMASTERDIR\java_uccs; C:\cloverleaf_dev\cis6.1\integrator\clgui\lib\cljava.jar; C:\cloverleaf_dev\cis6.1\integrator\java_uccs; Path=C:\cloverleaf_dev\cis6.1\integrator\site\bin; C:\cloverleaf_dev\cis6.1\integrator\site\scripts; \$HCIMASTERDIR\bin;\$HCIMASTERDIR\scripts;</pre> <p>These environment variables are set when <code>setroot</code> is called. Therefore, if you change the Master Site on the fly, you must run <code>setroot</code> to reset the variables.</p>
showroot	<p>This command is modified to show the Master Site if it is set:</p> <pre>HCI root is C:\cloverleaf_dev\cis6.1\integrator HCI master site is myMasterSite HCI site is case685</pre> <p>If it is not set, then this command prints the root and site info:</p> <pre>HCI root is C:\cloverleaf_dev\cis6.1\integrator HCI site is case685</pre>

Command-line tool	Description
hcirootcopy	<p>This command is modified to copy the mastersite entry in rootInfo from the source to the destination.</p> <p>If the entry is already set in the destination rootInfo, then the system prompts you for confirmation to override. If the <code>-f</code> option is given, then the system automatically overrides.</p>
Tcl search path	<p>Tcl resolves procedure references using the <code>\$auto_path</code> TCL variable, which is set to:</p> <pre>\$HciRoot\tcl\lib\cloverleaf \$HciRoot\tcl\lib\tfc \$HciSiteDir\tclprocs \$HciMasterSite\tclProcs \$HciRoot\tclprocs</pre>

Save the references of shared configurations into configuration files

The IDE saves the built-in resource names of the shared configurations without marking the corresponding configuration file. This is the same as saving local configurations.

The engine searches the configuration in the order: **Site > Master Site > Root > Master**.

Master is only for engine output configurations.

Root is used in Tcl procs, tables, and engine output configurations.

All other configurations only support the former two searching destinations.

RootInfo access control

The `rootInfo` can be accessed through the ACL/Role Manager.

If you have Read and Write permission or only Write permission on `rootInfo`, then you can open the **Root Preferences** dialog box to save any modifications.

If you have only Read permission on `rootInfo`, then you can open the **Root Preferences** dialog box but cannot do any modification.

If you have no Read and Write permission on `rootInfo`, then you cannot open the **Root Preferences** dialog box. Instead, an error dialog box opens, stating `Insufficient privileges to access resource`.

Data Encoding tab

An encoding option is on the Network Configurator, Testing Tool, and SMAT.

The default encoding for these dialog boxes is configured from the **Data Encoding** tab of the **Root Preferences** dialog box. The default is ASCII.

To configure the default encoding, select the encoding from the **Default Encoding** menu. This list contains all of the **Encoding List** entries. Entries can be added or removed.

When an encoding is selected from the **Default Encoding** menu and a new thread is created in Network Configurator, the new default encoding is shown in the property tab's **Encoding** field.

Note: ASCII and UTF-8 cannot be deleted.

You must restart the host server to update an encoding change.

Default encoding format

Use the default encoding format to customize the default encoding format selection for SMAT message viewing. This is configured in **Options > Root Preferences**.

When the selected message content is opened in **SMAT Database**, the system attempts to use the encoding format that is in the `ecd` file. If the file does not exist, or there is no valid value set in the file, then the system uses the default encoding format setting.

File transfer

The **File Transfer** dialog box is for remote users of client-only installations to connect to the host server. This is accessed in the IDE by selecting **File > File Transfer**.

After this is finished, those users can manipulate the files on the remote machine.

You can copy files to a fixed directory on the server called `shared_files`. This directory resides within the root. A status dialog box opens during the operation showing progress. It includes the file name and the percentage of the operation that is complete and remaining.

The `shared_files` folder is the top-level folder from which users manage files and folders.

The `shared_files` directory is automatically created the first time you use File Transfer.

Users cannot manage files and folders outside of the `shared_files` root.

Note: Only one file copy operation can happen at a time. Multiple users can transfer files to the server simultaneously and manage separate folders on the server.

Transferring data

- 1 In the **File Transfer** dialog box, select **File > Transfer File**. This opens a file selection dialog box.
- 2 Select the `shared_files` sub-folder in which to place the transferred file.
- 3 Specify the local file to transfer to the selected `shared_files` sub-folder and click **OK**.
- 4 Click **Start** on the status dialog box.

Selecting the environment

- 1 In the IDE, select **Server > Change**. A confirmation message is displayed.
- 2 Click **Yes**. The **Select a Server and Environment** dialog box is displayed.
- 3 Users have the option of selecting another environment from which to work.
The bottom pane lists the available environments on the host machine. Select an environment and click **Apply**. Client users can access any listed environment immediately without any halt in production. It is not necessary to stop and restart any tools or the host server.

Creating a new directory

- 1 Select where to place the new folder.
- 2 In the **File Transfer** dialog box, select **File > New Folder**. The **Create New Folder** dialog box is displayed.
- 3 Specify the new folder name.
- 4 Click **OK**. The new directory folder displays in the directory structure.

Moving a folder or file

- 1 Select the file to move.
- 2 In the **File Transfer** dialog box, select **File > Move Folder/File**. The **Select Move Destination** dialog box is displayed.
- 3 Highlight where to move the file and click **OK**.

Renaming a folder or file

- 1 Select the file to rename.
- 2 In the **File Transfer** dialog box, select **File > Rename Folder/File**. The **Rename File/Folder** dialog box is displayed.
- 3 Specify the new name and click **OK**.

Cloverleaf reconnect

When a host server connection is lost, you can do an emergency save and restoration procedure.

This feature is for connections opened by a configurator for the purpose of working on a configuration file through the host server.

A client cache stores temporary copies of configuration data. This is intended only for storing a single copy of the at-risk configuration data per disconnected session.

Emergency save

This feature is for connections opened by a configurator for the purpose of working on a configuration file through the host server. It supports and extends the client/host reconnect feature, giving you the option of performing an emergency save when the server connection is lost.

The Emergency Save feature does not compromise the current security and authentication models. You cannot work with a secured IDE or stand-alone GUI without authentication. With this feature, you cannot initiate the IDE or a stand-alone GUI in a disconnected mode. It can be used only in situations where the IDE or a stand-alone GUI was initiated in a connected mode and then lost its connection.

In situations where a reconnect is not possible, Emergency Save temporarily caches unsaved configuration data on the client side. In a later session the cached configuration data can be saved to the server.

The purpose of Emergency Save is to provide an emergency save procedure and not to provide a general client-side or back up process.

When a host server connection is lost, the **Server Connection Lost** dialog box opens, giving you the opportunity to cache unsaved data.

Clicking **Emergency Save** opens the **Emergency Save** dialog box, where you select the data to temporarily save on the client.

An emergency save is possible only when the configurator has lost its connection to the host server.

After an emergency save, you are returned to the IDE or stand-alone GUI in a disconnected mode. The connection must be made before restoration can happen.

The Emergency Restore Wizard opens when a connection is made or restored and there is cached data to save to the connected host server. It lists the cached files and the current lock owner of the cached host server file, if any. It also gives you the opportunity to select the files to restore. Un-restored files are removed with the Emergency Restore Wizard, or left in the cache for restoration at a later time.

Performing an emergency save and restoration

When changes are made to Configurator data and the connection is lost, follow these steps for an emergency save and restore:

- 1 When a host server connection is lost, the **Server Connection Lost** dialog box opens.

2 Click **Emergency Save.**

Check boxes are shown only for those configurations that were not saved when the host server connection was lost. If all configurations were saved when the host server connection was lost, then **Emergency Save** is unavailable.

3 Select the files for emergency save.**4 Click **OK**.** The selected files are now temporarily cached on the client side.**5 The **Emergency Save** dialog box closes and the **Server Connection Lost** dialog box remains open until **Reconnect** is clicked and the connection is restored.****6 After the connection is restored, the Emergency Restore Wizard opens. This lists the files that have been previously stored to the client cache through an emergency save.****7 Click **Next**.** The previously cached files are shown. Select the files to restore.**8 Click **Next**.** The Wizard shows the successfully restored files.**9 Click **Next**.** Any remaining files that have not been restored are listed with the option to remove them from the client cache. If all files are restored, then the next wizard step is skipped and the **Finished** dialog box opens. If no selections are made, then a prompt is given the next time you connect to the server.**10 Click **Next**.** The Wizard shows the number of files restored or removed.**11 Click **Done**.**

Reconnecting the client and host

When a host server connection is lost and any action is attempted that requires a host server connection, the **Server Connection Lost** dialog box opens. Use this dialog box to reconnect or perform an emergency save.

Click **Reconnect** to begin repeated attempts at host server reconnection. These attempts can be canceled at any time.

If the host server is restarted under a security mode, then the **Log on** dialog box opens before reconnecting. For security reasons, if the host server has been significantly reconfigured, reconnection may not be possible.

When reconnection is completed, the message changes from `Disconnected` to `Reconnection Successful`.

This feature is required to follow security protocol and does not compromise the current authentication model. This prevents users from communicating with the host server outside of the authentication process. This is performed by checking cached information or re-prompting users to log in as part of the reconnection.

If the security authentication fails, then the reconnection is stopped and the application ends.

Client cache

The client cache stores temporary copies of configuration data. This is intended only for storing a single copy of the at-risk configuration data per disconnected session. The cached data is only accessible by the user who saved it.

After a configuration that is stored in the cache is successfully saved to the server, the copy in the cache is deleted.

Cached data is stored in the `/integrator/client/cache` directory, which is created as required.

Cloverleaf ini settings

The `client.ini` file has these sections:

- `general`
- `host_hostname_audit_srv`
- `user_username`
- `user_username_xlate_config`
- `user_username_audit`
- `logs`

The `server.ini` file has these sections:

- `general`
- `firewall`
- `exports`
- `logging`
- `audit`
- `email`
- `security`
- `smat_search`

The `security.ini` file has these sections:

- `firewall`
- `security`
- `logging`
- `events_autopurge`

Boolean flag to enable or disable the auto purge events table feature. The default is `true`.

- `events_purge_history_duration`

Number of days at which entries older than this value in the events table are deleted after the timer task is run. By default, this is 7 days.

- `events_purge_history_interval`

Timer schedule interval. The default is 8 hours.

server.ini

These tables list the parameters of the `server.ini` file.

[general] section

Name	Options	Install Value	Install Config	Required	Description
jvm_args		-Xmx256	all host	TRUE	This is used for JVM arguments. The JVM heap size defaults to 256MB.
csc_monitoring_enabled	<p>true: CSC monitor server is started if value is true and Global Monitor server is enabled.</p> <p>false: CSC monitor server is not started if value is false or Global Monitor server is disabled.</p>		all host		Specifies that the CSC monitor server is enabled or disabled.
csc_rmi_port			all host		Specifies the CSC RMI port (21222).

[firewall] section

This table lists the parameters in the [firewall] section:

Name	Value
xltdebug_server_use	<p>true: host server traffic</p> <p>false: Direct connection</p>

[exports] section

This table lists the parameters in the [exports] section:

Name	Install value	Install config	Required	Description
environs	drive:cloverleaf\cis version\integrator\siteProto	all host	TRUE	This lists the valid site locations for this host server. Unless a default site is created in the install, this defaults to the siteProto directory.

[logging] section

This table lists the parameters in the [logging] section:

Name	Install value	Install config	Required
Cloverleaf_server_category	info	all host	FALSE

Name	Install value	Install config	Required
Cloverleaf_server_level	brief	all host	FALSE
host_server_category	info	all host	FALSE
host_server_level	brief	all host	FALSE
log_rmi_calls	false	all host	FALSE
debug_ssl	false	basic, adv host	FALSE

[audit] section

This table lists the parameters in the [audit] section:

Name	Install value	Install config	Required	Description
auto_trim	true	all host	TRUE	Indicates if auto-trimming is running in the Audit Log.
auto_trim_count	16000	all host	TRUE	The number of records that the Audit Log permits before auto-trimming.

[email] section

This table lists the parameters in the [email] section:

Name	Description
server_name	email server name
protocol	SMTP/SMTPs
port	port number
timeout	time-out in seconds
auth_required	true/false
tls_required	true/false
sender_email_addr	email address of the sender
account_name	user account of email
account_password	password of user base64 encoding
administrator_email_addr	email address of the administrator

Name	Description
cert_expiration_notification	true/false
cert_expiration_time	the days before certification expiration notification

[security] section

This table lists the parameters in the [security] section:

Name	Install value	Install config	Required	Description
ticket_life	16	basic, adv host	TRUE	The number of hours that a cached log-in session is valid.
host_cert_chain	prompt	basic, adv host	TRUE	<p>Lists the certificate names in the host's certificate chain. This should be set to <i>hostname-clserver-cert.der</i>; <i>username-cert.der</i>; <i>hie-cert.der</i>.</p> <ul style="list-style-type: none"> <i>username</i> is the name of the customer as it shows in the customer CA certificate. <i>hostname</i> is the name of the install machine. <p>The host certificate is created during the install.</p>
host_private_key	prompt	basic, adv host	TRUE	Contains the name of the host private key. The format of this name is <i>enc-hostname-clserver-key.der</i> .
password	prompt	basic, adv host	TRUE	<p>This is the password for the host server certificate.</p> <p>The host certificates are created during the install. This is the password users specify for these certificates.</p>
security_server_host	prompt	adv host	TRUE	<p>This is the location of the security server. This value is a host name that the host machine recognizes in a DNS lookup.</p> <p>Often, this is the fully-qualified domain name (<i>security_host.hie.com</i>), but in some networks it may be only the machine name without the domain (<i>security_host</i>).</p>
security_server_used	true	adv host	TRUE	This indicates if a security server is being used for this host. It should be present only for security server (advanced) installs.

Name	Install value	Install config	Required	Description
security_anonymous	false	NoSec host	TRUE	<p>This indicates whether the host server should use anonymous SSL for communications.</p> <p>If it is false, then the application defaults to straight RMI.</p> <p>This is not valid for any install other than a No Security install.</p>
basic_security_enabled	true	basic, adv host	TRUE	<p>This is set to true if basic or advanced security is enabled. It should not be present for a no security install.</p>
customer_ca_file_name	prompt	basic, adv host	TRUE	<p>This is the name of the customer CA certificate (<i>username-cert.der</i>).</p>
customer_ca_key_name	prompt	basic, adv host	TRUE	<p>This is the name of the customer CA private key certificate (<i>enc-username-key.der</i>).</p>
remote_Cloverleaf_server			FALSE	<p>This is a boolean that indicates whether a host server should bind in remote objects to the RMI registry. This is for use by the system configuration GUI clients.</p> <p>This value defaults to true and can be set to false.</p>
upload_private_key			TRUE/FALSE	<p>This indicate whether the private key can be uploaded to server side.</p>
acl_shared_host	Host name where the ACL is shared. This is valid only when the current host server is under advanced security.			<p>Note: This is for clustered HA environments.</p> <p>This value makes it possible to use a single ACL to cover multiple host servers, so that the primary server and the failover server can share a single ACL.</p> <p>By default, this entry is not visible. The value is the name of the local machine. When you configure it to another host server, the permission authorization on security server respects the ACL defined on that host server other than the one defined on the local host machine.</p>

[smat_search] section

This table lists the parameters in the [smat_search] section:

Name	Default value	Description
scan_interval	10 minutes	Scan interval, in minutes, that the scan is to find out which SMAT files have been created/modified. Messages in these files are queued for indexing.
scan_thread_number	5	Maximum number of Java threads that are used for scanning.
indexing_thread_number	5	Maximum number of threads that are used for indexing SMAT messages.
index_sleep_time	400 milliseconds	Sleep duration that happens during indexing, in milliseconds.
index_num_before_sleep	200	After indexing the specified number of messages, the indexing thread turns to sleep and then wakes up later.
index_buffer_size	1179 * 1000	Buffer size that is used to cache SMAT data during indexing, in bytes.

security.ini

These tables list the parameters of the `security.ini` file.

[firewall] section

	Default value	Install config	Required	Description
rmi_registry_port	13021	security	FALSE	The TCP port number for the RMI registry to serve requests. This port must be the same as the registry port numbers of the host server and any system configuration GUI client. [security] section
security_server_default_port	empty	security	FALSE	This corresponds with the RMI Object Port text box on the Server Administration GUI. Entering a port in the text box "fixes" the port, so that the port does not change when the security server is restarted.

[security] section

	Default value	Install value	Install config	Required	Description
password		prompt	security	TRUE	[security] This is the password used to encrypt and decrypt the private key file of the security server. The install process creates a public certificate and encrypted private key file. The security server does not start if this property is not set or is set improperly,
security_certificate_chain	host-clsecurity-cert.der;hie-cert.der	Install creates: <i>host-name-clsecurity-cert.der</i> ; <i>user-name-cert.der</i> ; <i>hie-cert.der</i>	security	TRUE	<p>This is the semicolon-delimited list of certificates. They represent the certificate chain of the security server's server certificate, to the customer certificate authority's certificate, to the Infor certificate authority's public certificate. Set this to <i>host-name-clsecurity-cert.der</i>; <i>user-name-cert.der</i>; <i>hie-cert.der</i>.</p> <ul style="list-style-type: none"> <i>username</i> is the name of the customer as it shows in the Customer CA certificate. <i>hostname</i> is the name of the security install machine. <p>The security host certificate is created during the install.</p>
security_private_key	enc-host-clsecurity-key.der	Install creates: <i>enc-host-name-clsecurity-key.der</i>	security	TRUE	<p>This is the file name of the encrypted private key file for the security server. The naming convention for this file is <i>enc-hostname-clsecurity-key.der</i>.</p>

[logging] section

	Default value	Install config	Required	Description
Cloverleaf_security_server_log	security.log	security	FALSE	This is the file name for the security server log file.
Cloverleaf_security_server_category	INFO	security	FALSE	This is the logging category for the main application of the security server and is used for filtering log messages. Valid values are PRODUCT, ERROR, WARNING, INFO, and DEBUG.
Cloverleaf_security_server_level	BRIEF	security	FALSE	This is the logging level for the main application of the security server and is used for filtering log messages. Valid values are SILENT, BRIEF, and VERBOSE.
security_server_category	INFO	security	FALSE	This is the logging level for the remote security server object and is used for filtering log messages. Valid values are PRODUCT, ERROR, WARNING, INFO, and DEBUG.
security_server_level	BRIEF	security	FALSE	This is the logging level for the remote security server object and is used for filtering log messages. Valid values are SILENT, BRIEF, and VERBOSE.
security_store_category	INFO	security	FALSE	This is the logging category for the system Security Store and is used for filtering log messages. Valid values are PRODUCT, ERROR, WARNING, INFO, and DEBUG.
security_store_level	BRIEF	security	FALSE	This is the logging level for the system Security Store and is used for filtering log messages. Valid values are SILENT, BRIEF, and VERBOSE.
log_rmi_calls	false	security	FALSE	This is a flag to turn on RMI logging in the Sun RMI layer. Options are true or false.
rmi_log	security_rmi.log	security	FALSE	This is the file name for RMI log messages.
debug_ssl	false	security	FALSE	This is a flag to turn on SSL logging in the Phaos API. Options are true or false.

client.ini

These tables list the parameters of the `client.ini` file.

[general] section

	Default value	Install value	Install config	Required	Description
fontsize		11	client installs	FALSE	This controls the base font size. Larger font sizes are calculated based on this value. If it is not present, then it defaults to 11.
debug	true	true	client installs	FALSE	When true, this indicates that unhandled exceptions are shown in the process output dialog box presented by the IDE. When false, the IDE does not show unhandled exception messages in the process output dialog box.
rmi_registry_port	13021	x	client installs	FALSE	This is the RMI registry port of the current host server. This port is used when attempting connections to all host servers.
host_connect_timeout_secs	15	15	client installs	FALSE	This is the maximum amount of time to spend on each interrogation of a system host when looking for a host server. There are currently three types of servers: RMI, ANON, and AUTH. When a user attempts to connect to a new system host, the client attempts a lookup on each server type for the amount of time that is specified by this parameter until successful.
jvm_args		-Xmx256m	all host	TRUE	JVM heap size defaults to 256MB

[host_hostname_audit_srv] section

This table lists the parameters in the [general] section:

	Install value	Install config	Re- quired	Description
url	//host:port/Cloverleaf_server	client installs	FALSE	This is the URL of the server that was last successfully connected to by a GUI. The section name varies depending on the host name. Each host has a section. The port may vary depending on the port number of the RMI registry on the target host server. This setting makes faster subsequent connection attempts to a known host server. This eliminates the interrogation usually required when connecting to a host for the first time.

[user_username] section

	Install value	Install config	Required	Description
certs	public_certificate1;public_certificate2;...	client installs	FALSE	This is the fully-qualified, semicolon-delimited list of public certificate files that have been used on the client machine to successfully log in to a host server.
private_keys	private_key1;private_key2;...	client installs	FALSE	This is the fully-qualified, semicolon-delimited list of private key files that have been used on the client machine to successfully log in to a host server.
hosts	host1; host2; ...	client installs	FALSE	This is the comma-delimited list of servers to which the client has had a successful connection.
last_host	host	client installs	FALSE	This is the last host to which a connection was successfully made. The name is used to access items in the host_<hostname> sections.
last_cert	public_certificate	client installs	FALSE	This is the fully-qualified path of the last certificate that was used for a successful connection to last_host.
last_private_key	private_key	client installs	FALSE	This is the fully-qualified path of the last private key that was used for a successful connection to last_host.
last_remote	true/false	client installs	FALSE	This indicates whether the last successfully connected host (last_host) was a remote or local connection.

	Install value	Install config	Required	Description
last_environ	environ_string	client installs	FALSE	This is the last system environment that was selected for the last host (last_host). This value is found in the Select a Server and Environment dialog box.
login_user	user name	client installs	FALSE	This is the user name of the last user that logged on successfully. It is used to determine the section names starting with user_username.
use_cred_cache	true/false	client installs	FALSE	This indicates whether you must store successful connections to host servers locally. This eliminates the requirement for the user to log in to each GUI on start-up. The credential cache does have a distinct lifetime set by the administrator in the server.ini file (see the ticket_life parameter in server.ini). When the cache expires, the user is prompted to log in.

[user_username_xlate_config] section

	Install value	Install config	Required	Description
app_size	width height	client installs	FALSE	This is the space-separated last width and height of the Translation Configurator main frame.
app_pos	x y	client installs	FALSE	This is the last x-y position of the Translation Configurator main frame.
input_dlg_visible	true/false	client installs	FALSE	This indicates whether the Input File Format dialog box was open or closed when the user last shut down the Translation Configurator.
input_dlg_size	width height	client installs	FALSE	This is the space-separated last width and height of the Input File Format dialog box when the user last shut down the Translation Configurator.
input_dlg_pos	x y	client installs	FALSE	This is the last x-y position of the Input File Format dialog box when the user last shut down the Translation Configurator.

	Install value	Install config	Required	Description
output_dlg_visible	true/false	client installs	FALSE	This indicates whether the Output File Format dialog box was open or closed when the user last shut down the Translation Configurator.
output_dlg_size	width height	client installs	FALSE	This is the space-separated last width and height of the Output File Format dialog box when the user last shut down the Translation Configurator.
output_dlg_pos	x y	client installs	FALSE	This is the last x-y position of the Output File Format dialog box when the user last shut down the Translation Configurator.
temp-var_dlg_visible	true/false	client installs	FALSE	This indicates whether the Temporary Variables dialog box was open or closed when the user last shut down the Translation Configurator.
temp-var_dlg_size	width height	client installs	FALSE	This is the space-separated last width and height of the Temporary Variables dialog box when the user last shut down the Translation Configurator.
temp-var_dlg_pos	x y	client installs	FALSE	This is the last x-y position of the Temporary Variables dialog box when the user last shut down the Translation Configurator.

[user_username_audit] section

	Install value	Install config	Required	Description
num_view_entries	x	client installs	FALSE	This is the number of Audit Log entries that the user requires to view in the Audit Log Viewer.
receive_live_updates	true/false	client installs	FALSE	This indicates whether the user requires to view Audit Log messages in real time as they are posted to the Audit Log. When false, the user can refresh the Audit Log view manually by clicking Refresh in the Audit Log Viewer.

[logs] section

	Install value
logfile	Not used
log_directory	This is for specifying the log path of a client. If this property is not set, then the default is <i>%HCI ROOT%/client/logs</i> .

Cloverleaf wizard

The wizard is a browser-based user experience that walks you through a workflow. This workflow creates an interface from scratch or using existing BOX (Buildable Object eXchange) templates.

Note: The Translation Configurator in the Wizard is read-only.

The wizard is installed as part of Cloverleaf, with all required HTML5 resources. It is accessed by a standard URL on the Cloverleaf host server.

A Cloverleaf thread is referred to as a “system” in the wizard.

Objects that are not supported by the wizard editor, such as translations, are included in the project and deployed with the project.

A project can be created or edited in the wizard configuration.

- Data formats
- Routing rules
- Lookup tables
- Global variables
- Translations

Cloverleaf BOXes that are transferred to the remote host server are deployed using the standard Cloverleaf BOX management features. The user interface is browser-based. Wizard projects are created that contain artifacts such as:

Server interface

The user interface is browser-based. Wizard projects are created thatThe server side is a RESTful interface whose data format is JSON. See [Cloverleaf API](#).

If the server RESTful API is disabled, then the wizard prompts an error message, indicating the wizard services were disabled by the administrator.

There is no effect on performance of the engine at runtime; however, you may require additional disk storage for Cloverleaf history artifacts.

An active projects' objects are stored in a Cloverleaf site. `hcirootcopy` captures these directories on system upgrades.

When a wizard project is packaged, the server stores Cloverleaf BOXes that contain all the artifacts of the defined interface at `HCIR00T/box`.

Additional functions

With lookup tables, you can create, edit, or view Cloverleaf lookup tables, with an additional option to test the table configuration.

Global variables can be created, edited, or deleted.

Each option, project, system, data format, route, lookup table, and translation has context training available. This is an option on the main page.

Editable parameters

You can create systems and edit parameters for these protocols:

Protocol	Configurable parameters
TCP/IP PDL	Port number Hostname
TCP/IP	Port number Hostname
Web Service	URL modifications for: Ports Path
Fileset/Local	Directory path
Fileset/FTP	Directory path Username Password Hostname

Creating and editing a project

You can create a new project using the wizard, or you can use a previously configured BOX.

A list of all BOXes on the remote host server is provided. You can select a BOX on you remote host server to deploy into a project.

You can also save projects to a BOX.

A project can be edited in the wizard configuration. You can add, edit, or delete individual artifacts within the project configuration.

Setting up the wizard

To set up wizard using the host server:

- 1 Install Cloverleaf Integration Services.
- 2 Set-up basic security.
- 3 Check all of the options in the **Server Administration > Web Server** tab.
- 4 Restart the host server.
- 5 Open a browser and go to: `https://hostname:15057/clwizard`.
Use the same log-in **User Name** and **Password** as your security install.

Auto-Start scripts

The Auto-start script UI is a tool for configuring auto-start scripts. This is controlled by `application/hcserver admin` in advanced security mode. There is no access control in basic security mode. By default, the settings are stored in `autostart/as.hci.profile`.

If `server/auto-start.profile` is present, then it has higher priority than the first one.

If `server/auto-start.profile` is added after editing the auto-start scripts, then the host server must be restarted to make the read/write location point to the new location.

Note: For additional information on auto-start functionality, see [Auto-Start](#) on page 1158.

Auto-start features and options:

Feature	Description
Projects	<p>Projects are controlled by auto-start scripts. When all projects are selected, they are always controlled by the auto-start scripts.</p> <p>It is unnecessary to update auto-start scripts for new projects.</p>
Excluded processes	<p>By default, all project processes are started by auto-scripts. Projects added to this list are excluded from auto-start.</p>
Projects repository file system	<p>When deploying CIS to the production environment, some projects that are created on replicated or shared file systems are linked to <code>HCROOT</code>.</p> <p>When auto-start projects are linked outside <code>HCROOT</code>, you must set the "Projects repository file system". The auto-start scripts ensure that the link exists when the scripts start.</p>

Feature	Description
Automatically start the host server on Windows	<p>This option applies in Windows systems. The existing NT service starts the host server.</p> <p>Setting this variable instructs the auto-start scripts to restart the host server.</p>
Start projects by a single process	<p>In most cases, auto-start scripts create multiple processes to start a configured project, one process for one project. This causes a failover. Having many projects start up in parallel exhausts the system resources and causes unexpected errors.</p> <p>In these instances, select Defragment Database. When this is selected, projects are started one-by-one. This saves system resources, but runs slower. The single process mode saves logs on Windows at <code>HCIROOT/server/fake_stdout.txt</code> and <code>fake_stderr.txt</code>.</p> <p>On Linux, a project's start-up log is created in sequence at <code>/var/log/message</code>.</p>
Defragment Database	<p>Running the <code>hcidbdefrag</code> command can be helpful when you have a large number of messages in recovery. However, you must exercise caution.</p> <p>For example, if you have 200K messages, then the process takes many minutes, or hours, to complete. This option allows you to take control of the action, to avoid an unnecessarily long process time.</p>

SAML(Ming.le)

The admin user can configure SAML(Ming.le) on the **Admin Web** user interface.

Parameters and options include:

- **Enable SAML(Ming.le) user authentication**
This enables/disables the Cloverleaf IDE SAML(Ming.le) user authentication. The default is disabled.
- **CA password**
This is used in creating the user certificate and private key files.
If the CA password already exists, then it is not necessary to re-enter.

There are two metadata and properties sections on the interface:

- IDP properties and metadata
- SP properties

IDP properties and metadata:

- **Entity ID**
This IDP (Identity Provider) property corresponds to the `idp.saml.entityid` key in the IDP properties file.
Click **Import IDP Properties** to import the Entity ID from the IDP properties file.
- **Domain name**
This IDP (Identity Provider) property corresponds to the `idp.last.activity.domain.name` key in the IDP properties file.
Click **Import IDP Properties** to import the domain name the from the IDP properties file.
- **Cookie name**
This IDP (Identity Provider) property corresponds to the `idp.last.activity.cookie.name` key in the IDP properties file.
Click **Import IDP Properties** to import the cookie name from the IDP properties file.
- **Home page URL**
This IDP (Identity Provider) property corresponds to the `idp.mingle.homepage.url` key in the IDP properties file.
Click **Import IDP Properties** to import the home page URL from the IDP properties file.
- **Portal URL**
This IDP (Identity Provider) property corresponds to the `idp.mingle.portal.url` key in the IDP properties file.
Click **Import IDP Properties** to import the portal URL from the IDP properties file.
- **Metadata**
This is the IDP (Identity Provider) metadata file content in XML format.
Click **Import IDP Metadata** to import the metadata from the IDP metadata file.

SP properties:

- **Entity ID**
This is a required property of the SAML SP (Service Provider). This must be unique across the deployment region.
Copy this from the registered Ming.le service provider.
- **SSO URL**
This is an internal URL generated by CLAPI for SAML implementation.
- **SLO URL**
This is an internal URL generated by CLAPI for SAML implementation.

Scheduler Log

Click the **Scheduler Log** menu to open the **Scheduler Log** page. This menu is located at **Admin Tools > Server Configuration**.

Configurable parameters are:

- **Level**
Sets the scheduler log level. This value can be "Info" or "Debug".
- **Automatic Log cycling**
Enables/Disables log cycling.
- **Threshold in kilobytes**
Sets the threshold for cycling. This is a non-negative integer with a range of 0-999999999.
This option is only available when **Automatic Log cycling** is enabled.
- **Enable log history**
Enables/Disables log history. This option is only available when **Automatic Log cycling** is enabled.
- **Delete the oldest log history file when history files number exceeds n.**
Sets the maximum history file number. This is a non-negative integer with a range of 0-999999999.
This option is available only when **Enable log history** is enabled.
- **Delete the oldest log history file when history files total size exceeds n**
Sets the maximum history file size. This is a non-negative integer with a range of 0-999999999. This option is available only when **Enable log history** is enabled.

These settings are read from and save into `hcischeduler.ini` file.

Click **Save** to save the settings.

Configuring localization

Localization is a means of changing the language of the wizard application.

Note: "Localization" and "user profile" are not in the initial release.

All text labels for the user interface are stored in a resource file that is read by the Server and used for presentation. The user profile contains a setting for selecting the resource file to use. This localizes the user interface.

In localization, all text labels for the user interface are stored in a resource file which is read by the server and used for presentation.

The supplied `resources-en-us.json` file (English) is used as the template for the new language. This contains all the keys/values for the words.

- 1 Copy this file and rename it to your language, for example, `resources-us-language.json`.
- 2 Update the line `selected-language-id: en-us` to reflect the new language by changing **en-us**.
- 3 Then, change the words to the new language.
- 4 After that, change `config.json` to use the new resource file.

Infor Ming.le® integration

You can configure the SSO authentication settings using the **Server Administration**. These settings are only active if the server is configured to use LDAP for user management.

The CL Wizard web user interface supports Infor Ming.le integration.

CLAPI has been modified to meet the requirements for Infor Ming.le integration certification.

All external modules have been upgraded.

The CLWizard integrated with Infor Ming.le requires the "Infor IFS System" role to map to the "Cloverleaf administrator" role when the application starts. The Cloverleaf "administrator" role is available after upgrading to advanced mode on Security Server.

"cladmin" is the administrator role for Cloverleaf, and must be manually predefined on Security Server before using the Infor Ming.le portal.

For example, when the Infor Ming.le role name is "Infor-SystemAdministrator", it maps to "cladmin" by default.

Infor Ming.le language selection

The Wizard supports the Infor Ming.le language profile. You can select your preferred language in Infor Ming.le. The preferred language setting is used by the Wizard application. English is the default language for the Wizard if an unsupported language is selected in Infor Ming.le.

Preferred languages are:

- English (default)
- Simplified Chinese
- Traditional Chinese

Infor Ming.le bookmarks

The Wizard supports the use of Infor Ming.le bookmarks. You can add a specific site interface object (NetConfig, Lookup Table, and so on) as a Infor Ming.le bookmark. Selecting the Infor Ming.le bookmark logs you into the Wizard and opens the selected object's tool/view.

Infor Ming.le homepage widget

The Wizard supports the use of Infor Ming.le homepage widgets. The Wizard's homepage widget is a view/list of the Wizard projects (sites) that are available to you on the server.

Infor Ming.le drillback messages

The Wizard supports drillback messages. These messages are similar to Infor Ming.le bookmarks. With drillback messages, you can share a view of a specific interface object. You can also send a drillback message of a specific interface object (Lookup Table, Translate Config, and so on). The recipient of the message clicks on the message and is logged into the Wizard. The view is set to the specific object referenced in the drillback message.

Translations

On **Projects**, select a project and click the **Translations** menu to open the **Translation** page. This page contains a data grid that displays all translation files. On this page, you can configure the actions and test the current translation file.

The URL is: `/#/project/{siteName}/xlt/list`.

Note: The master site translation files do not display.

Name is the translation file name and link to the Edit/View Translation page.

When you click a column header, the translation files are sorted and filtered.

On the toolbar:

- **Site Doc** opens the **Site Documentation** page from CIS.
- **Test** opens the **Translation Test** dialog box.
- **Save** is enabled when you click the **Move Up/Move Down/Remove/Activate/Deactivate** button.

The **Move** buttons are enabled when one or more continuous rows that share the same parent are selected.

The **Activate/Deactivate** buttons do not display in these conditions:

- The selected row is a child of a deactivated parent row.
- The selected row is an Else action and the above If action is a deactivated row.

Action Detail dialog box

Click **Detail** to go to the **Action Detail** dialog box. This displays the same action data that is listed on the **Site Document**.

Click the link in the dialog box to go to the corresponding translation/table page for the **Include** and **Table** actions.

Translation actions

All defined translation actions are listed on the Translation Action table.

Click **Detail**, or the operation action name, to open the **Action Detail** page.

On this page, **Description** displays the action's source and destination values. For example, `"0(0).N3(0).#1(0) -> _implicitRules.extension(0).valueInteger"`. The content of the action is a comment.

Translation Test dialog box

This dialog box displays the test screen for translation files.

These actions are available:

Field/Action	Description
Enter data message	Specify the translate file data.
Choose data file	Select the data file from <code>\$HCISITEDIR/Translate/</code> .

Field/Action	Description
Process all records/Process one record	<p>Click Process all records for the system to read the selected data file and process all messages in it.</p> <p>Click Process one record for the system to read the selected data file and process only the first message in the file.</p>
Show field names	Select to display field names.
Leak detection	Select to run memory leak detection.
Print unreadable characters	Prints unreadable characters without changing to hex, and instead prints in readable print.
Detail level	<p>Select the detail level for the test output that is reported in the Result pane.</p> <p>Detail levels go from 0 (raw, unparsed data) to 4 (most detail).</p>
Line termination format	<p>Select the format in which the test messages are saved:</p> <ul style="list-style-type: none"> • "Newline Terminated" reads the data in the file until it finds a newline character, makes all the data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser. • "Length Encoded" reads the first 10 characters to determine the length of the first message, reads that many characters into a message, and then sends it to the parser. • "EOF Terminated" reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.
Encoding	Select multi-byte encodings. When an encoding is selected, an <code>-e</code> encoding option is added to the corresponding command line.
Result	Displays the test result after clicking Run Command .
Run Test	<p>Click this to run the command and then display the command output in the Result text area.</p> <p>If the test fails, then error information displays in the Result text area.</p> <p>This keeps the last setting information to the local session and displays the information when the dialog box is reopened.</p>

Adding new variants

On **Projects**, select a project and click the **Variants** menu. This contains a data grid that displays all variants.

To add a new variant:

- 1 Click **Add New Variant** to open the **Add Variant** dialog box.
- 2 Click **Create** to open a list of available messages. Click **Add Message** to open the **Add Message** dialog box.
- 3 Do the same for segments, fields, and so on.
- 4 Click **Test** to verify. This uses the CLAPI.
- 5 Click **Enrich** to compare your new message with a message definition.

NetConfig metadata

Additional fields are required for the wizard that are not natively supported by NetConfig. The key-value pairs, called NetConfig metadata, are:

- `systemName`: This is used to provide a more descriptive name than the thread name; for example, “ADT feed from Registration.” Level: Thread
- `systemVendor`: This is used to provide a name for the vendor associated with the remote system; for example, “EPIC.” Level: Thread

This is an example of how it looks in the NetConfig, located within a protocol definition: `{ META {{"desc":"new system file","vendor":"Epic"}} }`

Examples

These examples demonstrate the use of the Infor Cloverleaf wizard.

Use case	Description
Deploying from an existing BOX	<p>To deploy an interface using existing Cloverleaf artifacts previously developed by Cloverleaf interface developers:</p> <ol style="list-style-type: none"> 1 Create a project and download the Cloverleaf BOX as the starting point. 2 Modify the thread names so that they do not interfere (as duplicates) with existing interfaces on the Cloverleaf server. 3 Change the port and IP address. 4 Test the changes using the Cloverleaf wizard. 5 Manage the changed interfaces with Global Monitor.

Use case	Description
Data model builder	<p>To create a new HL7 V2 variant using test data supplied from a vendor:</p> <ol style="list-style-type: none"> 1 Create a project. 2 Select the data model builder. <p>This guides in creating and testing the HL7 data model.</p>
Lookup table	<p>To create a Cloverleaf lookup table from data supplied from a vendor in a CSV format:</p> <ol style="list-style-type: none"> 1 Create a project. 2 Select Lookup Table. <p>This guides in creating and testing the lookup table.</p>
Route configurator	<p>To create a route for ADT^A18 merge transactions though another translation:</p> <ol style="list-style-type: none"> 1 Select an existing project and then select the Route Configurator. This guides in how to add the ADT^A18 to an existing route that does not have this TrxID. 2 Create a new route between two existing systems for ADT^A18 and apply the translation to the route. 3 After configuring the route, use the Testing Tool to test the routing configuration.
Interface end-to-end	<p>This creates an ADT interface between two systems. Events A01, A02, and A03 are to be routed raw. Event A08 is to be routed using an existing translation provided in a Cloverleaf BOX.</p> <ol style="list-style-type: none"> 1 Create a new project based on the BOX. The Cloverleaf wizard starts a workflow which walks you through the process. 2 Using the wizard, create two new systems that use the HL7 MLLP protocol. 3 Create a route between the new systems that route ADT^A01, A02, A03 raw, and A08 using the translation. 4 Test the configuration. 5 After testing, package the configuration as a BOX for deployment to production.
Complex interface with expert assistance	<p>To build a complex interface between two systems that route order and result transactions bi-directionally:</p> <p>In this workflow, translations are required that have not been pre-built.</p> <ol style="list-style-type: none"> 1 Using the Cloverleaf wizard workflow, create a framework with systems, protocol configuration, and raw routing. 2 Package the project into a BOX and send to an implementer who can create and add translations to the configuration.

Use case	Description
Collaboration with Infor Ming.le Drill Back Messages	<p>To confirm the contents/mapping of the Location Lookup Table in a new ADT Interface with the Business Manager:</p> <ol style="list-style-type: none"> 1 Select the Location lookup table and select Share with. 2 Route to the Business Manager in the dialog box and type a note asking for a review of the table. 3 The Business Manager receives the message and clicks the Table link. 4 The Business Manager logs into the Wizard and the view is set to the Location table. 5 The Business Manager reviews the table.
Infor Ming.le Homepage Widget	<p>To continue work on a new interface:</p> <ol style="list-style-type: none"> 1 Log into your Infor Ming.le account. This opens the Infor Ming.le homepage that contains the Cloverleaf Wizard widget. This widget contains an icon for every visible wizard project on the server. 2 Select your <code>dev</code> site from the widget. This logs you into the Wizard and opens the default page for the <code>dev</code> site.

CLWizard/CLAPI site/BOX configuration workflow

In these example topics, the work flow is to BOX a Cloverleaf site and deploy the site to another Cloverleaf environment.

This is accomplished through the CLWizard/CLAPI or the IDE. For details, see:

- [Configuring a CLWizard/CLAPI site/BOX using CLAPI and CLWizard](#) on page 150
- [Configuring a CLWizard/CLAPI site/BOX using the IDE](#) on page 153
- [Creating a BOX template](#) on page 154

Configuring a CLWizard/CLAPI site/BOX using CLAPI and CLWizard

These steps are for configuring a CLWizard/CLAPI site/BOX using CLAPI and CLWizard.

- 1 Create the BOX site.
 - a On the **Project** page, select the site to open.
 - b On Project home, click **Save As Template**.
 - c On the **Save Project As Template** page, specify the template name and click **Save**. Wait until the new BOX is created.
 - d On the **Templates (BOXes)** page, confirm that the newly created BOX is listed.

- Using the API:

POST: `https://HostName:15057/clapi/api/command/hcibox?s=SiteName&boxname=BoxName`

- Request URL:

`https://localhost:15057/clapi/api/command/hcibox?s=helloworld&boxname=helloworldbox`

- Request headers:

```
{
  "Accept": "*/*"
}
```

- Response body:

```
{
  "status": 0,
  "stdout": "The BOX 'helloworldbox' has been created successfully.\r\n",
  "stderr": ""
}
```

Note: The Wizard does not support adding additional library files into the BOX. These are added using the IDE.

2 Export and download the BOX to your local machine.

- Export a BOX to a directory under `HICROOT` using an API:

POST: `https://HostName:15057/clapi/api/command/hcibox?e=PathToExportBox&boxname=BoxName`

- Request URL:

`https://localhost:15057/clapi/api/command/hcibox?e=temp%2Fhelloworld&boxname=helloworldbox`

- Request headers:

```
{
  "Accept": "*/*"
}
```

- Response body:

```
{
  "status": 0,
  "stdout": "The BOX 'helloworldbox' has been exported to 'temp/helloworldbox.box' successfully.\r\n",
  "stderr": ""
}
```

- Downloading the BOX zip package using an API:

Note: For Infor Cloud Cloverleaf 19.1.2, the Wizard and API do not support binary file downloading.

Beginning in CIS 20.1, there is a new CLAPI interface that supports downloading binary files that are under a site directory. To do this, use:

```
GET: /clapi/api/root/site/SiteName/downloadfile?file=FilePath
```

3 Upload the BOX from your local machine to Cloverleaf.

Use this API to upload a BOX file to the HClROOT/temp directory:

```
POST: /api/root/uploadtempfile
```

```
// curl to get CSRF token
curl -G https://<HostName>:15057/clapi/api/security/csrf -k --header "Authorization:Basic
Base64 Encoded UserName:Password"
  -c cookieFile.out -v
// curl to upload BOX to remote cloverleaf HClROOT/temp directory
curl https://HostName:15057/clapi/api/root/site/helloworld/uploadfile?"file=BoxName&ap
pend=false"
  -k --header "Authorization:Basic Base64 Encoded UserName:Password" -H "Content-Type:mul
tipart/form-data"
  -F "inbinfile=@FullPathOfTheBOXToUpload" --header "X-CSRF-TOKEN:CSRF Token" -b cook
ieFile.out -v
```

4 Import the BOX, using:

```
POST: https://HostName:15057/clapi/api/command/hcibox?i=BoxFileRelativePath&boxname=BoxName
```

- Request URL:

```
https://localhost:15057/clapi/api/command/hcibox?i=temp%2Fhelloworldbox.box&boxname=hel
loworldbox2
```

- Request headers:

```
{
  "Accept": "*/*"
}
```

- Response body:

```
{
  "status":0,
  "stdout": "The BOX 'temp/helloworldbox.box' has been imported successfully.\r\n",
  "stderr": ""
}
```

5 BOX deployment can be to an existing site or a new site.

To deploy the BOX to an existing site:

```
POST: https://HostName:15057/clapi/api/command/hcibox?d=SiteName&v=true&boxname=BoxName
```

- Request URL:

```
https://localhost:15057/clapi/api/command/hcibox?d=helloworld&v=true&boxname=helloworldbox2
```

- Request headers:

```
{
  "Accept": "*/*"
}
```

- Response body:

```
{
  "status":0,
  "stdout": "Deploying BOX 'helloworldbox2' to site 'helloworld'\r\nCopying resources to
box 'helloworldbox2'. . . . .
  "stderr": ""
}
```

To deploy the BOX to a new site:

- In CL Wizard, select the BOX on the **Templates (BOXes)** page and click **Create New BOX**.
- or
- POST: `https://HostName:15057/clapi/api/command/hcibox?c=NewSiteName&v=true&boxname=BoxName`

- Request URL:

```
https://localhost:15057/clapi/api/command/hcibox?c=helloworld2&v=true&boxname=helloworldbox2
```

- Request headers:

```
{
  "Accept": "*/*"
}
```

- Response body:

```
{
  "status":0,
  "stdout": "Deploying BOX 'helloworldbox2' to site 'helloworld2'\r\nCopying resources
to box 'helloworldbox2' . . . . .
  "stderr": ""
}
```

Configuring a CLWizard/CLAPI site/BOX using the IDE

To configure a CLWizard/CLAPI site/BOX using the IDE:

- 1** BOX the site.
 - a** Log in to the IDE and connect to the remote Cloverleaf host.
 - b** Open Remote Commands and run:

```
hcibox -s SiteName BoxName
```

- 2 Add additional library files to the BOX.
 - a Open BOX Manager and double-click the BOX to modify.
 - b On the **BOX Property** dialog box, open the **Resources** tab.
 - 1 Click the triangle icon located on the bottom left corner.
 - 2 On the pop-up menu, click **Add Libraries**. This opens the **Add Additional Libraries** dialog box.
 - 3 Click **Add Folder** or **Add File**. This adds to the list the target folders/files that are located under the Cloverleaf `HCIR00T`.
 - 4 Click **OK**. This applies your selection and closes the **Add Additional Libraries** dialog box.
 - 5 Click **Save** on the **BOX Property** dialog box to finish.
- 3 Export and download the BOX to your local machine.
 - a Log in to the IDE and connect to the remote Cloverleaf host.
 - b Open BOX Manager and right-click the BOX to be exported. On the pop-up menu, click **Export**.
 - c On the **Export BOX** dialog box, specify the location at which to save the BOX on your local machine. Optionally, you can specify a password to encrypt the BOX.
 - d Click **Save**. The BOX is then downloaded to the specified location.
- 4 Upload BOX from your local machine to Cloverleaf.
 - a Log in to the IDE and connect to the remote Cloverleaf host.
 - b Open BOX Manager and right-click in the tool's content area. On the pop-up menu, click **Import**.
 - c On **Import BOX** dialog box, specify the BOX from your local machine. If the BOX is encrypted, then you must enter the password.
 - d Click **OK**. The BOX is then uploaded and imported to the target Cloverleaf host.
- 5 BOX deployment can be to an existing site or a new site.

To deploy the BOX to an existing site:

 - a Log in to the IDE and connect to the remote Cloverleaf host and site.
 - b Open BOX Manager and right-click the BOX. On the pop-up menu, click **Deploy**.
 - c To complete the deployment, follow the Wizard's **Deploy BOX to Site** steps.

To deploy the BOX to a new site:

 - a Log in to the IDE and connect to the remote Cloverleaf host.
 - b Open Remote Commands, and run this command:

```
hcibox -c SiteName BoxName
```

Creating a BOX template

To create a BOX template:

- 1 Go to **CLWIZARD > Projects** and click a project.
- 2 On the project view page, select and check the threads on the thread list.
- 3 Click **Save Selected As Template**. This opens the **Save Selected Threads As Template** dialog box.
- 4 Select the template options and click **Save**.

Cloverleaf API

Applications can access (R/W) Cloverleaf configurations using the Cloverleaf RESTful API interface. This API is a web application deployed in the embedded Apache Tomcat within the Cloverleaf host server.

To enable the Cloverleaf RESTful API, the host server must be running in security mode.

The user interface is a RESTful web service exposed in JSON format. This service is exposed using the existing host server port.

Detailed configuration of the embedded Tomcat service is provided through the Tomcat configuration file.

There is no effect on engine performance at runtime; however, users might require additional disk storage for Cloverleaf history artifacts.

The server stores all of the copies of artifacts created by these processes and maintains revision controls to ensure that no Cloverleaf objects are overwritten. The API provides support for objects under Cloverleaf version control.

The password in the request URL for lookup table and global variable should be base64encoded. The user name and password for Authorization should be in the format "username:password," and the whole content should be base64encoded.

RESTful API

The Cloverleaf API uses a RESTful web service architecture and JSON (JavaScript Object Notation) as the data-interchange format. You have the ability to create, edit, and delete Cloverleaf objects that were previously only available in the Cloverleaf GUI.

The API is configured on/off through the Server Administration tool.

To comply with HIPAA rules and regulations, all activity requests, adds, edits, views, and updates are written to the audit log with user details.

There is no effect on engine performance at runtime; however, you might require additional disk storage for Cloverleaf artifact history.

All administrative functions are available through the API.

Javascript SDK is delivered with HTML documentation for each API.

The API respects the Cloverleaf security model (None, Basic, or Advanced with ACL).

For details, see [Best Practices : REST API](#).

Common issues for CLAPI usage

This example contains many of the common issues for CLAPI usage.

When the Lookup Table API returns configuration details from an advanced table lookup, it has been reported that it works when the table is basic. An advanced lookup table does not return configuration details.

Usage:

Currently, all the .tbl file configuration details can be saved and retrieved through:

- `/api/site/{siteName}/lookuptable/{tableName}/get`
- `/api/site/{siteName}/lookuptable/{tableName}/save`

These use the same class: `CloverTblConfig`.

Here are two different inputs:

- A basic lookup table instance with a SQL statement: How to distinguish the basic and advanced modes. The "dbTable" entry has marked that in basic mode, the input JSON data requires a "dbTable" key-value. For example:

```
{
  . . . .
  "dbConnection": "sqltest",
  "dbTable": "ibats_sqlserver",
  "dbsql": "SELECT Flag FROM ibats_sqlserver WHERE CreateDate=<CreateDate",
  "isProcedure": false,
  . . . .
  "procedureType": "0"
}
```

In advanced mode, it must not include this key. For example:

```
{
  . . . .
  "dbConnection": "ora_conn_10g",
  "dbsql": "{call ZOE_11312_SELECTMULTI(<i_a1, <i_a2, o_a3 OUT, o_a4 OUT, o_a5 OUT)}"
  "isProcedure": true,
  . . . .
  "procedureType": "1"
}
```

- This is an advanced table instance with a `StoredProcedure` type. To get the procedure type, use a SQL statement or stored procedure.

The `isProcedure` and `procedureType` keys determine whether the type is a SQL statement or stored procedure.

- If it is a SQL statement, then:
 - `isProcedure` : "false"
 - `ProcedureType` : "0"
- If it is a stored procedure, then:

```
isProcedure : "true"
ProcedureType : "1"
```

Using the API

These sections provide information on using the API securely.

Security

The service uses the host server and security server to validate and grant access that is based on user permissions.

To enable or disable the Cloverleaf API, on the **Server Administration > Web Server** tab, the **Enable Cloverleaf API** check box enables/disables the Cloverleaf API. The default is cleared (disabled).

- If enabled, then the server returns results on a request or a rejection notification.
- If disabled, then an error code (404) is returned to any clients attempting to access the API.

For security reasons, the CSRF token is added to the HTTP POST request to access the CLAPI.

Note: Regarding the HTTP POST request, except for the HTTP GET request, all others are affected, including POST, PUT, and DELETE.

For example, this is a curl command for sending a POST request with the CSRF token. To do this, you must send two requests:

- 1 Send the request to get the cookie file that contains the session ID. This returns the CSRF token value.

```
curl -G https://hostname:15047/clapi/api/security/csrf -k --header
"Authorization:Basic YWRtaW5pc3RyYXRvcjpQQHNzd29yZDAx" -c cookieFile.out -v
```

YWRtaW5pc3RyYXRvcjpQQHNzd29yZDAx is the Base64 encoded string for username:password.

2980a518-1728-46bc-a95d-9767baac2f48 is the CSRF token value in the response. This is later used in the request header.

- 2 Send the real request for starting the process. The CSRF token and cookie file are used to get the session ID that is being imported into the header.

```
curl https://hostname:15047/clapi/api/site/{sitename}/process/{processname}/
start -k --header "Authorization:Basic YWRtaW5pc3RyYXRvcjpQQHNzd29yZDAx"
-H --header "X-CSRF-TOKEN:2980a518-1728-46bc-a95d-9767baac2f48"
-b cookieFile.out --request POST -v
```

Access permission

For basic security access permission, the Cloverleaf administrator grants access to users through a log-in using a user name and password.

Advanced security access permission is required to assign the corresponding permission for CLAPI in the ACL/Role Manager tool. In addition to user name and password authentication, the service uses the security server to grant access based on the user's permissions.

- `clapi` under the integrator node on the ACL tree is used to control whether the user has the permission to use CLAPI.
- `usercmd` under the integrator/command node is used to control whether the user has the permission to run generic commands.
- `userfile` under the integrator/config node is used to control whether the user has the permission to upload generic files.
- Users should be given correct permissions on the ACL tree to access/modify any other resources.
- When creating a site through the Cloverleaf API, users should provide a pre-defined permission template on the security server. This is so that the corresponding permissions are given to the newly created site. Otherwise, the user should manually define the permission on security server after creating the site. Then, the user can do configuration on that site.

CSRF token

Cloverleaf API enables the CSRF token to avoid cross-site request forgery attacks. It requests that each invocation that is not a GET API invocation needs to have CSRF TOKEN in the header.

The CSRF token value is returned with JSON format by the response.

Users must invoke a new `/api/security/csrf` API with the session ID. This is saved in a cookie after log-in, to get the CSRF TOKEN for that session. This returns `{"csrf": "csrf_token_value"}`.

The CSRF token cookie is http-only. This indicates that Javascript cannot read the cookie.

CLAPI does not save the CSRF token into a cookie; instead, it provides an API to get the CSRF token.

The client calls the new API to get the CSRF token, caches it in memory, and sends it as X-CSRF-TOKEN with the request.

XSS attack protection

To avoid XSS attack, CLAPI decodes the request body when it receives a request from a client. It encodes the response body when it sends back the response to a client. Because the request and response body are almost JSON objects, CLAPI only provides encoding/decoding on these characters:

"&" <-- "&"

"<" <-- "<"

">" <-- ">"

```
"" <-- "&#x27;"
"/" <-- "&#x2F;"
```

Use cases

A Cloverleaf software developer is asked to develop an application where users can add or modify Cloverleaf objects through a custom-built front-end. The developer uses the RESTful API to develop the front-end applications that use Cloverleaf objects.

Command line API example:

A user has a script for running on the server using an API. The user places the script in a specific directory on the site. The API is configured to only run user-defined scripts from this site. The user develops a Client-side utility that runs the script using the API and processes the output.

Command line API using Curl example:

A user has a script for running on the Server using an API. The user places the script in a specific directory on the site. The API is configured to only run user-defined scripts from this site. The user runs the Client-side utility that runs the script using the API through Curl and processes the output.

Adding Cloverleaf objects

When adding Cloverleaf objects, a software developer can create an application for users to add or modify Cloverleaf objects through a custom-built user interface.

For a command line API through Curl, if you have a script that is required to run on the server through an API, then you can:

- 1 Place the script in a specific directory on the site.
- 2 Configure the API to run only user-defined scripts from this site.
- 3 Run the client-side utility that runs the script through the API using Curl to process the output.

Examples:

- ```
curl -G https://127.0.0.1:15057/clapi/api/sites/helloworld/alerts -k --header "Authorization:Basic dXNlckE6MTExMTEx" -v (dXNlckE6MTExMTEx is base64 encoded user:password authentication string "userA:111111")
```
- ```
curl https://127.0.0.1:15057/clapi/api/root/preferences -k --header "Authorization:Basic dXNlckE6MTExMTEx" -H "Content-Type:application/json" --data-binary @data.json -v
```

Runtime tools

The runtime tools control and monitor connections, processes, and system information. The tools are:

- **Connection Dump**
Use this to check configuration items and compare thread configurations without using the Network Configurator.
- **Database Administrator**
This queries, manipulates, and changes the message transaction database. The database contains two states of messages: Recovery log and Error log.
Both transaction logs are created and maintained by the engine.
- **Error Database**
This contains all functions available in the SMAT Database, plus the ability to see error state and error context. Transactions can be edited and resent as they can with the SMAT Database.
- **Network Monitor**
This controls and monitors network connections.
All actions are saved to an engine output log and time-stamped. Use the log to monitor use and to debug connections.
- **Site Daemons**
Two site daemons, or background processes, are used in each site:
 - **Lock Manager (UNIX only)**
The lock manager controls access to the error and recovery databases.
 - **Monitor Daemon**
The monitor daemon monitors engine processes and threads.
- **SMAT**
Use this to configure to save all inbound and outbound messages processed by the engine. Use the saved messages to provide information for statistics or retransmission.
SMAT saves copies of messages only when a message has been successfully sent to or received from a connection. If delivery is unsuccessful, then messages are sent to the recovery or error databases.
- **SMAT Database**
This provides the option to enable encryption, and is also available in Global Monitor. These messages are accessed through the SMAT Database GUI.
- **Shell Window**
This provides access to the command line, where all utilities and processes are available.

- Site Daemons

Two site daemons, or background processes, are used in each site:

- Lock Manager (UNIX only)

The lock manager controls access to the error and recovery databases.

- Monitor Daemon

The monitor daemon monitors engine processes and threads.

Connection Dump

The Connection Dump reviews thread configurations without using the Network Configurator.

Use the Connection Dump to check configuration items and compare thread configurations.

Viewing the connection dump

You can review a thread configuration's details without running Network Configurator.

- 1 Click the **Process Name** arrow on the **Connection Dump** dialog box.
- 2 Select the process containing the thread to review.
- 3 Click **Apply**. Connection information is shown in the field.
See [Connection information](#).

Connection information

This table shows the information shown in the Connection Dump.

The remainder of the connection information reflects each connection's protocol configuration.

Parameter	Description
Process	Name of the engine process.
Connection	The thread name for which details are displayed.
EO Proc	Engine output (EO) configuration for the process. To configure, select Process > Configure from the Network Configurator.
EO Config	Engine output configuration for the connection. To configure, use the Properties tab of the Network Configurator.

Parameter	Description
EO Msg	Engine output configuration for the message originating from this thread. The message EO alias remains with the message as it travels through the engine. Its effect is in addition to any process or thread EO aliases. To configure, use the Properties tab of the Network Configurator.
Groups	Used to control or monitor threads in Network Monitor, and lists groups with which this thread belongs. To configure, use the Properties tab of the Network Configurator.
Data Type	Source of data, or record type, used for determining the transaction ID. To configure, use the Inbound tab of the Network Configurator. Click the Trx ID Determination Format arrow to select the format from the list. Then, click Edit to open the Trx ID Determination dialog box.
Options	Thread startup mode on process startup in Network Monitor. For automatic startup, select Auto-start Connection on the Properties tab of the Network Configurator.
RecoverDb	Recovery database that is used for this thread. To enable, select Use recovery database on the Properties tab of the Network Configurator.
Hold	Outbound data or reply messages that are held for the thread. To enable, select Hold on the Outbound tab of the Network Configurator thread connection menu.
SendOnly	Only outbound messages are processed by this thread; inbound messages are discarded. To configure, select Outbound Only on the Inbound tab of the Network Configurator.
Disk Msgs	Indicates if the Disk-Based Messages option is active.
Save Msgs (in)	Inbound messages are saved in this connection as they are read into the system. To enable, select Save inbound messages in the Inbound pane on the Properties tab of the Network Configurator.

Parameter	Description
Inbound File	File name for inbound saved messages. Two files are created using this base file name, one with an <code>.idx</code> extension and one with an <code>.msg</code> extension. To configure, select Save inbound messages in the Inbound pane on the Properties tab of the Network Configurator. Then, specify a file name in the File field.
Save Msgs (out)	Outbound messages are saved. To configure, select Save outbound messages in the Outbound pane on the Properties tab of the Network Configurator.
Outbound File	File name for outbound saved messages. Two files are created using this base file name, one with an <code>.idx</code> extension and one with an <code>.msg</code> extension. To configure, select Save outbound messages in the Outbound pane on the Properties tab of the Network Configurator. Then, specify a file name in the File field.
Comm Type	Protocol communication type. To configure, in the Properties tab of the Network Configurator, select Protocol . Then click Properties to configure the protocol.

Database Administrator

The Database Administrator processes messages from the transaction database. The transaction database is different than, and independent from, any saved message log file. Although each connection has its own saved message file, all connections use the same transaction database.

There are two types of transaction databases:

- Recovery database

The recovery database is used to recover messages that are in transit if any of these conditions exist:

- The engine is restarted
- The engine unexpectedly goes down

When the engine starts running after a failure, it automatically continues processing messages from the recovery database. If a connection does not have the recovery database option enabled, then messages in transit during an engine failure are lost.

Messages are only in the recovery database when they are in the system. They are removed as soon as they are successfully sent to the outbound connection.

Use Network Configurator during site configuration to enable recovery for specific threads.

- Error database

The transaction database also recovers messages from processing and translation errors. These errors happen when one of these conditions exist:

- A message is formed incorrectly
- Some stage of message processing fails

When an error happens, the message that caused the error is saved in the error database. The state describing the cause of the failure is also saved.

The majority of messages in the error database are undeliverable due to an error condition.

Error messages remain in the error database until manually removed.

Database Administration wizard

The default is the **Database Administrator Classic** dialog box.

From this dialog box, there are two configuration choices:

- Use the **Database Administrator** dialog box for configuration.
- Click **Wizard**, located at the bottom left of the **Database Administrator** dialog box, to use the Wizard.

The Database Administration wizard is a configuration aid that guides you through the entire configuration process. When the wizard is running, it is shown as a modal dialog box.

After opening the wizard, click **Next** and follow the prompts.

Database Administrator definition

This table shows the options for processing messages from the transaction database.

Option	Description
Recovery Database	Processes messages in the recovery database.
Error Database	Processes messages in the error database. When this is selected, clicking Edit/Resend opens the Edit/Resend Error Database Message(s) dialog box. After the necessary changes have been made, click Resend Current Message or Resend Marked Messages to open a dialog box. On the dialog box, select whether to resend to the Engine Queue or to a File .
Screen	Sends output to a new dialog box.
File	Saves the selected messages to a file for future processing. If a file is not listed, then output displays in a new dialog box.

Option	Description
Normal Format	Click this when not saving messages to an output file. This provides basic message information. This option is available only when Screen is selected.
Long Format	Click this when not saving messages to an output file. This adds details useful in sorting. This option is available only when Screen is selected.
Ultra-Long Format	Click this when not saving messages to an output file. This is useful in debugging Tcl procs. This option is available only when Screen is selected.
Binary Length-Encoded	Select this for resend operations. Binary length-encoded, the default, is the most portable file format because it does not use special characters to delimit records. End binary length-encoded file names with <code>.bin</code> . This option is available only when File is selected.
Newline Terminated	Use this option when selecting a single message for manual editing, where each record is separated from adjoining records by a newline character. Do not use this format when the data within the message contains one or more newline characters. There are no restrictions on file names, except that they should not end in <code>.bin</code> . This option is available only when File is selected.
Filename	Specify the file name in which to save selected messages. Click Folder to select an existing file from the file browser. To specify a new file that is binary length-encoded, specify the <code>.bin</code> extension; do not specify the <code>.bin</code> extension if the file is newline terminated. This option is available only when File is selected.
Overwrite Existing File	Select to overwrite the output file. This option is available only when File is selected.
Database Order	Click the arrow to open a list of sorting-order choices.

Option	Description
Map data	<p>Select to convert data through the named character data mapping table. Specify the table name in Filter. Most common mapping options are ASCII to EBCDIC, and EBCDIC to ASCII. There are four standard character maps that come with the system:</p> <ul style="list-style-type: none"> • <code>ibm_a2e</code> • <code>ibm_e2a</code> • <code>hcl_a2e</code> • <code>hcl_e2a</code> <p>In addition, you can define your own character maps.</p>
Destination	<p>Select to select the destination connection. Then, click the arrow for a list of connections that are defined in the current Network Configuration file.</p>
Delete Selected Messages	<p>Removes unnecessary messages. Use this option to prevent messages from being resent through a another query.</p>
Do Not Delete Selected Messages	<p>Resends messages.</p>
Source	<p>Select to choose the source connection from a list of connections that are defined in the current Network Configuration file.</p>
Owner	<p>Select to choose the connection that currently owns the message from a list of connections that are defined in the current Network Configuration file.</p> <p>As messages move through various processing stages, other threads "own" the message. The thread that owns a message is responsible for the recovery of that message at system restart time.</p> <p>This attribute should be used only at the direction of Support personnel.</p>
Message Type	<p>Select to choose a message type, Data Type or Reply Type.</p>
Message State	<p>Select to indicate the stage of processing for recovery messages or the reason for failure on error messages.</p>

Option	Description
Message ID	<p>Select to list the internal system message ID, as assigned by the engine as messages flow through the system. Use only the starting or ending message ID for an inclusive and open-ended message search.</p> <p>For correct results, a database record is accessed by one of these methods:</p> <ul style="list-style-type: none"> Making one row of entries, Start or End, and leaving the other as 0.0.0 (or blank). Making both rows of entries, Start and End, identical. <p>When making entries into the Start or End row:</p> <ul style="list-style-type: none"> Preceding fields of 0 or blank are stripped from the command-line argument: <pre>blank blank 1 == 1 0 0 1 == 1 0 2 1 == 2.1 blank 3 4 == 3.4</pre> <ul style="list-style-type: none"> A non-zero field causes all subsequent fields to be interpreted as at least 0: <pre>1 blank blank == 1.0.0 1 blank 2 == 1.0.2 1 0 3 == 1.0.3 0 2 5 == 2.5 blank 3 blank == 3.0</pre> <p>Use this only at the direction of Support personnel.</p>
Date Range	Specifies a date range in which to check or delete messages before/after a certain date in the error/recovery database.
Message Count	Retrieves the message count of the error/recovery database.
Grep	Filters the output. This is available when Error Database is selected.
Verbose Mode	Select to enable verbose mode.
Show Error Details	When Error Database is selected, selecting this check box shows details from the error database when you click Apply Command .
Show Readable Message Flags	Shows an expanded readable view of the flags metadata field.

Option	Description
Wizard	Opens the configuration wizard. Use the wizard as a guide through Database Administrator configuration. See Database Administration wizard .
Show Command	Opens the Command dialog box. Use this dialog box to view the command that runs when Apply Command is selected.
Apply Command	Click to check the requested selection. This opens the Result dialog box. If the request is correct, then the selection criteria creates the output file or shows the result. Sometimes, when a command is applied and Ultra Long Format is selected, the output information might show output information longer than the dialog box. In these cases, there is a Word Wrap check box located at the bottom of the dialog box to aid in reading these messages.
Edit/Resend	This function is available when Error Database is selected. This opens the Edit/Resend Error Database Message(s) dialog box. Click Resend Marked Messages to open a dialog box for configuring the resend parameters. Click Apply to resend the marked messages.

Message and error states

This table shows the recovery database message states:

- | | |
|--|------------------------------------|
| 1: On IB pre-TPS queue | 2: On IB post-TPS queue |
| 3: Sent by ICL to xlate | 4: Staged at ICL received by xlate |
| 5: On pre-xlate queue | 6: Currently in xlate processing |
| 7: On post-xlate queue | 8: Sent by ICL to destination |
| 9: Staged at ICL received by destination | 10: On OB pre-TPS queue |
| 11: On OB post-TPS queue | 12: On FWD pre-TPS queue |
| 13: On FWD post-TPS queue | 14: Successful delivery |
| 15: Failed delivery | 16: OB reserved for IB reply |
| 17: OB prewrite | 18: Unbacked queue |
| 19: On Java IB protocol | 20: On Java OB protocol |

This table shows the error database message states:

100: Failed to get Trx ID	101: Unsupported Trx ID
103: Tcl failure in Trx ID determination	104: No destination for reply
105: No destination for message	106: Disallowed Gateway routing
107: Incorrect remote ICL server	108: Incorrect dest to receive
See State 107 note	EDB_ROUTE_DESTNOTMATCH
200: Failure in IB Reply TPS eval	201: Failure in IB Data TPS eval
202: Failure in OB Reply TPS eval	203: Failure in OB Data TPS eval
204: Failure in FWD Reply TPS eval	205: Failure in FWD Data TPS eval
300: XLT config file unloadable	301: Internal XPM Tcl failure
302: Tcl callout error	303: Tcl callout abort
304: XPM data retrieval failed	305: XPM data store failed
306: XPM bulkcopy operation failed	307: Input data validation failure
308: Output data validation failure	309: XPM default value retrieve failed
310: XPM math operation error	311: \$xlateOutVals unset
312: XPM numeric comparison error	313: XPM string comparison error
314: XSLT transformation failure	
400: Failure in protocol startup	401: Failure in ACK/NAK proc
402: Failure in SENDOK proc	403: Failure in SENDFAIL proc
404: Failure in REPLYGEN proc	405: Retry count exceeded
406: Tcl failure in startup SendOk TPS	407: Tcl failure in startup SendFail TPS
408: Tcl failure in protocol driver TPS	409: Tcl failure in pre-write TPS
410: Failed to recover reserved outbound message	414: User code error in UPoC protocol read TPS
415: User code error in UPoC protocol write TPS	416: Inbound encoding conversion error
	See State 416 note
417: Outbound encoding conversion error	418: Unable to deliver to specified server connection
419: Bad data (error) in message writing	
500: Internal failure: Unable to read message data chain	501: Error database TPS error
1000: Internal failure: Start route	1001: Internal failure: Bad route config

1002: Internal failure: No route config

1003: Internal failure: Bad route detail

1004: Internal failure: Bad format

State 107 note

Error database state 107: EDB_ROUTE_REMOTEICLDEST (Incorrect remote ICL server) happens when an inter-site routing message, recovery database state 7 RDB_XLATE_POST_XLATE, is being recovered from the start-up of the engine. If both the remote ICL server (ICL destination) are unresponsive and the ICL destination definition is changed in NetConfig, then the message is moved to the error database with state 107.

The message is put into the error database during recovery from the recovery database if both of these conditions are met:

- The message cannot be delivered to the port it thinks it should go to.
If this is the case, then the message stays in the recovery database and in the queue because the server might be unresponsive.
- The configuration, as loaded when the process/thread is started, no longer matches the port the message expects.
If this is the case, then the message stays in the recovery database because the configuration has changed. The message can be delivered, so that delivery continues when possible.

During the error database resend, the port is corrected as the newly configured port.

The destination, including port number, is set when a message is routed, not when the message is delivered.

State 416 note

A message with this error state indicates an error happened during encoding from the encoding setting to UTF-8 when it is received in the inbound protocol thread. For example, the encoding setting of the inbound thread does not match the real message. In this situation, the message keeps its original encoding in the SMAT.

When resending such a message from SMAT, the engine converts it from the `-e` option that is given by the resend command to the encoding setting of the inbound thread. It then puts the message in the engine queue. In the engine queue, the message is treated as a new message from the inbound thread. The engine does the encoding conversion from the encoding setting of the inbound thread to UTF-8 again.

If the resend command does not give the `-e` option, then the engine skips the first encoding conversion. It directly puts the message from SMAT to the engine queue.

Use one of these methods to fix the encoding error and resend the error message:

- All the messages coming to the inbound thread have another encoding from the thread setting. The thread setting is incorrect.
Stop the thread and change the encoding setting to the correct one. Then, start the thread and resend the error message with/without the `-e` option.
If `-e` is given, then the encoding argument is the same as the message.
- Only the error message has another encoding from the thread setting. Other messages coming to the thread still have the same as thread setting.

Resend the error message with the `-e` option without changing the thread properties. The encoding argument is the correct one of the message.

Database Administrator command-line options

Some configuration options are available through the command line. Select any options to tailor your view of messages in the database.

Caution: Do not terminate `hcidbdump`! Termination before it is finished corrupts the database.

Usage examples:

```
hcidbdump [-U userid] [-v] -B [tarfile]
```

```
hcidbdump {-r|-e} [-U userid] [-l] [-L [-x]] [-v] [-D] [-f source_system] [-o owner_system] [-d dest_system] [-t type] [-s state] [-S time] [-E time] [-M map] [-O time] [-a] [-b [eof]] [-F] [-n metadata_filename] [-m mid] [-c] [-C] [-R] [-w searchExp] [output_file]
```

- `-B` performs database backup.
- `tarfile` specifies the file name where the database backup is written. If no `tarfile` is specified, then `tar` is written to `$DBFPATH/cldb.tar`.
- `-r` saves message from recovery database.
- `-e` saves message from error database.
- `-U userid` specifies the user ID to access database.
- `-l` specifies long form output. This is ignored when `output_file` is specified.
- `-L` specifies ultra-long form output. Overrides `-l`.
- `-x` gives an expanded view of flags. This is ignored unless using `-L`.
- `-v` specifies verbose operation.
- `-D` deletes messages after processing.
- `-f source_system` selects messages with a specified source.
- `-o owner_system` selects messages with a specified owner.
- `-d dest_system` selects messages with a specified destination.
- `-t type` selects messages with a specified type:
 - `D` specifies data.
 - `R` specifies reply.
 - `U` specifies unknown.
- `-s state` selects messages with a specified state.
- `-S time` selects messages at or after a specified date/time.
- `-E time` selects messages at or before a specified date/time.
- `-M map` applies the map to each message. Uses the `msgmapdata` command.
- `-O time` orders output by ascending date. Time flags are:
 - `i` specifies inbound arrival time.
 - `o` specifies outbound time.
 - `r` specifies recovery time.
 - `x` specifies xlate start time.

- `-a` appends data to output file.
- `-b [eof]` dumps message to output file in length-encoded or end-of-file format.
 - `-b` dumps in length-encoded format.
 - `-b [eof]` dumps in end-of-file format. Argument `[eof]` is optional.

You must provide an output file name for `-b [eof]`; otherwise, an error message results.

- `-F` deletes sent messages without prompting for confirmation.
- `-n metadata_filename` enables metadata to be dumped from the error database into a file of the necessary format.

See [Resending with modified metadata](#).

- `-m mid` selects messages with specified message IDs:
 - `x.y.z` specifies only this message.
 - `x.y.z1:x.y.z2` specifies `x.y` messages `z1` through `z2`.
 - `-m` overrides all other restrictions.
 - `-m` does not work with `-o`.
- `-c` shows context from error database.
- `-C` prints only the message count.
- `-R` is recovery mode for the recovery database.
- `-w searchExp` searches the message content with the specified expression. The search is case-sensitive, and works only for the Error Database.
- `output_file` specifies the file name where messages are written. If no output file is specified, then message data is written to stdout.

All specified constraints are ANDed together.

Disk-based messages

The recovery database stores disk-based messages. Make disk-based messages available at the process level. The overall memory limit, MSGSPACELIM, is specified in megabytes. It indicates the amount of memory that the process allocates for message data before storing messages to the recovery database. If the MSGSPACELIM field is 0, then no memory is allocated.

A memory leak happens if:

- One or more outbound protocol threads go down.
- **Disk-based Queuing** for the process was not selected.

To prevent this leak from causing the system to cease operating, the post-translate queue depths and virtual memory usage are monitored.

This is configured in the Network Configurator's **Process Configuration** dialog box.

Recovery database message flags

- MSG_FLAG_KEEPPONDISK

Set when a message is created. This is used to keep message data on disk, in the recovery database, instead of in memory (RAM). It operates the same way as using the recovery database for recovery purposes; after being set, it remains in effect until the message leaves the engine, or when message delivery is successful.

- MSG_FLAG_USERDB 0x00008000

Used to store messages for recovery purposes.

After a message is marked for recovery, it is backed by the recovery database throughout its life in the engine.

The recovery database backs up the thread and process queues. If an engine process or thread stops running, then it retains the message data from the recovery database when the thread or process is restarted.

- MSG_FLAG_ISONDISK 0x00001000

The message is kept on the hard disk, and not in the recovery database.

Configuring disk-based messages

- 1 Select **Process > Configure** in the Network Configurator to open the **Process Configuration** dialog box.
- 2 Select **Disk-based Queuing**. This enables **Threshold in Megabytes**.
- 3 In **Threshold in Megabytes**, specify the maximum number of megabytes the log file can contain before cycling.

If disk-based queuing is selected and the total required RAM for the queues exceeds this threshold, then all subsequent data posted to the queues is saved to the recovery database.

After the RAM requirements return to below this value, the message data is stored in RAM.

Disk space

The system databases reuse disk space when messages are removed.

These databases use disk space up to the maximum size of the file system. As a message is removed from a database, that file's disk space remains open to store another message. As more messages are added to the database, the disk utilization increases.

To restore database disk space and make it available for other messages:

- 1 Clean all records in the recovery and error databases.
- 2 Initialize the databases and control files.

Cleaning databases

The recovery and error databases sometimes require cleaning at the site level. When cleaning the databases, check to ensure the messages are not required.

Note: Descriptions of inbound and outbound are from the engine's point-of-view.

Using the recovery database

- 1 Shut down all inbound threads in all processes for the site.
- 2 Specify `hcidbdump -r`. This returns the contents of the recovery database.
- 3 Write this script to a file (UNIX):

```
#!/usr/bin/tcsh
while (1)
  hcidbdump -r |wc -l
  sleep 10
  clear
end
```

- 4 Prepare the script for running with `chmod +x filename`.

When the script returns 8, there are no messages in the recovery database. Eight (8) is the number of lines included in the header and an empty report.

You must `setroot` and `setsite` to the correct site before running the script. On Windows platforms, there is only one database per root. On UNIX platforms, there is one database per site.

Using the error database

- 1 Shut down all inbound threads in all processes for the site.
- 2 Specify `hcidbdump -e`. This returns the contents of the error database.
- 3 Specify `hcidbdump -e -D -F` to delete the messages in the error database.
To delete the messages in the error database and save them to a file, specify:

```
hcidbdump -e -D filename
```

Initializing databases

Note: This procedure destroys all data in the recovery and error databases. Save all important files. Data may be lost if the records in the recovery and error databases have not been processed.

Initialize the database to reclaim disk space. When messages are removed from the database, there is no disk space reclaimed; new messages reuse the same space.

- 1 Shut down all running engine processes at the site.
- 2 Stop the Lock Manager using `hcisitectl -k l`.
- 3 Use `hcisitectl` to verify that the Lock Manager is stopped.
- 4 Specify `hcidbinit -A -C` to initialize the databases and the control files. A warning prompts with `This is an extremely destructive operation! Are you sure?`
- 5 Specify `y` to initialize the databases and control files.
- 6 Restart the Lock Manager.

Error database tool

The Cloverleaf engine saves error messages in a SQLite database, to provide more ease of use and flexibility for users. The Error Database tool manages the error database for message search.

This is used similar to the SMAT Database tool, where you can do an HL7 smart search on the error database.

The Error Database tool contains all functions available in the SMAT Database tool, along with the ability to show the error state and error context. Transactions can be edited and resent, similar to the SMAT Database tool.

The auto-complete suggestion for the error database uses the same format/alias databases as the SMAT Database. The aliases that are configured in the **Site Preferences** dialog box, or the fields that are added/created in HL7 and HPRIM, are also shown in the error database.

Previously, if a message was sent to the error database, the user had no control over the processing of that event. The Error Database tool gives you a control point before the message is written to the error database. You can change the error message content or route the message to another context.

For example, you can:

- Change the message content to add information before the message goes to the error database.
- Stop the message so it does not go to the error database.
- Send a NAK to the source to notify there is an error when processing the message.
- Save the message to file or another external source, and resend the message that is based on it.
- Change the message and redo the step that cause the error.

This tool is similar to the SMAT Database, with these differences:

- There is no file tree list, since there is only one database for storing error messages.
- Support is provided for the error-based metadata: state, last recovery state, error info, class, and owner thread.

You can search using these metadata as part of the search criteria. The error state and last recovery state metadata are displayed in the search result by default. The error info, class, and owner thread are optional to display. This is specified in the **Customize Columns** dialog box.

- The **Resend** dialog box is the same as the one in the Database Administration tool.
- You can delete messages using the **Delete** buttons.

One button is located on the search result toolbar. When messages are marked in the search result, the button is enabled.

Another button is on the message viewer toolbar.

When **Delete** is clicked, a prompt opens for confirmation. After deleting, the current search is refreshed.

For information on encrypting/decrypting the error database, see [hcidbcrypt](#).

The `e\log.e\logdb` SQLite error database file is located in `$HCISITEDIR/exec/databases`.

The criteria definitions of error messages are saved in `criteria_def_username.ini` files. This is the same as the SMAT Database tool, except that these files are stored under `%HCISITE%\exec\databases\ criteria_defs`.

Tcl keyed list

In the NetConfig file, the error database configuration can be configured as a Tcl keyed list in the process or protocol level. When it is configured in the protocol level, it overwrites the configuration in the process level.

```
{ process processname
  { ERRDB_TPS
    {PROCS {{ARGS {}}} {PROCS hcierrdbhandle} {PROCSCONTROL ENABLED}}}
    {MAXRETRYTIMES 0 }
  }
}
{ protocol protocolname
  { ERRDB_TPS
    {PROCS {{ARGS {}}} {PROCS hcierrdbhandle} {PROCSCONTROL ENABLED}}}
    {MAXRETRYTIMES 0 }
  }
}
```

The error database configuration can also be configured in the `siteInfo` file for the site level.

```
erbtps_procs={ ERRDB_TPS {{ARGS {}}} {PROCS hcierrdbhandle} {PROCSCONTROL ENABLED}} }
errtps_max_retrytimes=0
```

Command line usage

This table shows the available commands for the error database.

You can also use `hciCmd` from a command-line interface to resend from the error database. This feature is available from the Database Administrator GUI.

Command	Usage
hcidbdump	<p>This supports a regex search on message content and dump error context.</p> <pre>hcidbdump {-r -e} [-U userid] [-l] [-L [-x]] [-v] [-D] [-i] [-f source_system] [-o owner_system] [-d dest_system] [-t type] [-s state] [-S time] [-E time] [-M map] [-O time] [-a] [-b [eof]] [-F] [-n meta_file] [-m mid] [-c] [-C] [-R] [-w searchExp] [Output_file]</pre> <p><code>[-w searchExp]</code> supports regular expression search on message content.</p> <p><code>[-c]</code> displays context from error database.</p>
hcidbinit	<p>This initializes the SQLite error database file.</p> <pre>hcidbinit [-A] [-r] [-e] [-i] [-m [-s threadName]] [-t] [-C] [-f]</pre>

Command	Usage
hcidbsetvers	<p>This gets/sets the error database versions from the SQLite error database file.</p> <pre>hcidbsetvers [-0] [-f] [-q] [-u userid] [-v]</pre>
hcidbconvert	<p>This converts messages in the current site's Raima database to the SQLite database.</p> <pre>hcidbconvert -e [-v] [-p password] [targetDbFile]</pre> <p><code>-e</code> is the error database.</p> <p><code>-v</code> is verbose.</p> <p><code>-p password</code> sets the password for the SQLite database.</p> <p><code>targetDbFile</code> converts the messages into the specific SQLite <code>targetDbFile</code> file.</p> <p>For error database conversion, by default <code>hcidbconvert</code> exports messages from Raima database files under <code>\$HCISITEDIR/exec/databases</code> and imports messages into <code>\$HCISITEDIR/exec/databases/elog.sqlitedb</code>.</p>

Error database schemas

Error database schemas are stored in these tables:

- **elog_info**
- **elog_msgs**

The **elog_info** table contains this content:

Field Name	Field Type	Description
1 Version	INTEGER	Error database schema version
2 ECD	VARCHAR	Currently, this is UTF-8

The **elog_msgs** table contains this content:

Field Name	Field Type	Description
1 Class	INTEGER	0:UNDEFINED 1:PROTOCOL 2:ENGINE
2 DataFmt	VARCHAR	Data Format

Field Name	Field Type	Description
3 DataLen	INTEGER	Message Content Len
4 DestConn	VARCHAR	Destination Threads
5 DriverCtl	VARCHAR	Driver Control
6 EdbState	INTEGER	Error Logging State
7 ErrorString	VARCHAR	Error Context
8 Flags	INTEGER	Flags
9 GroupID	INTEGER	Group ID
10 GroupMidDomain	INTEGER	Group Message ID
11 GroupMidHub	INTEGER	Group Message ID
12 GroupMidNum	INTEGER	Group Message ID
13 MessageContent	BLOB	Message Content
14 MidDomain	INTEGER	Message ID
15 MidHub	INTEGER	Message ID
16 MidNum	INTEGER	Message ID
17 MidsAreNull	INTEGER	NULL mid flag
18 OrigDestConn	VARCHAR	Original Destination Threads
19 OrigSourceConn	VARCHAR	Original Source Thread
20 OwnerThread	VARCHAR	Owner Thread
21 Priority	INTEGER	Priority
22 RdbState	INTEGER	Recovery Logging State
23 Retries	INTEGER	Message Number Retries
24 SepChars	VARCHAR	User Defined Separators
25 SkipXlt	INTEGER	Skip Xlate
26 SourceConn	VARCHAR	Source Thread
27 SrcMidDomain	INTEGER	Source Message ID
28 SrcMidHub	INTEGER	Source Message ID
29 SrcMidNum	INTEGER	Source Message ID
30 TimeIn	INTEGER	Inbound Start Time
31 TimeOut	INTEGER	Outbound Start Time

Field Name	Field Type	Description
32 TimeQCur	INTEGER	Current Queue Start Time
33 TimeQTot	INTEGER	Total Queue Time
34 TimeRec	INTEGER	Last Recovery Update Time
35 TimeXlt	INTEGER	Xlate Start Time
36 Type	INTEGER	0:UNDEFINED 1:DATA 2:REPLY
37 UserData	VARCHAR	User Data
38 XltThread	VARCHAR	Xlate Thread

Error trap TPS

When an error message is transitioned to the error database, you can control the event processing through the error trap TPS. You can specify procs to be called before messages are sent to the error database.

For example, you can show, save, stop, resend, or NAK, the message before it goes into the error database.

Configuration locations include:

- Network Configurator:

The error TPS can be configured in the protocol thread and process levels. The NetConfig is:

- Process level:

```
process helloworld {
  { ERRDBTPS {
    { ERRTPSPROCS {{ARGS {}}} {PROCS cl_show_err_msg} {PROCSCONTROL ENABLED}}}
    { RETRIES 2 }
  } }
}
```

- Protocol thread:

```
protocol in {
  { ERRDBTPS {
    { ERRTPSPROCS {{ARGS {}}} {PROCS cl_show_err_msg} {PROCSCONTROL ENABLED}}}
    { RETRIES 2 }
  } }
}
```

- Engine Output Configurator:

EO module dbi:etps

- Database Administrator:

501: Error database TPS error

- TPS disposition:

A RETRY disposition is in the `DispositionList` of Tcl and Java UPoC.

- **Error database TPS:**

The error database TPS gives you a control point before a message is written to the error database. This means you have the opportunity to change the error message content and to route the message to another context.

For example, you can:

- Change the message content to add information before the message goes to the error database.
- Stop the message so it does not go to the error database.
- Send a NAK to the source to notify there is an error when processing the message.
- Save the message to a file or another external source, and resend the message that is based on it.

Samples and templates include:

- `errdbtps.tcl`
- `errdbtps_template`
- `%HCIR00T/tclprocs`

TPS mode message flow

These TPS processing modes indicate the timing and scenarios when the engine calls the corresponding Tcl code fragment:

- **TPS_MODE_START**

When the thread has started, but is not yet scheduled to run and process events, the engine calls the thread initialization function. This is the function in which it calls the DBI initialization function for error and recovery database setup. During DBI initialization, it sets up DBI thread specific data (DTD data). In this stage, it gets error database TPS specific data from the NetConfig file. It then sets the TPS context in the xlate thread data and protocol thread data. Then, it runs the TPS start mode.

- **TPS_MODE_RUN**

When the thread is processing an event and it goes to an error, it calls a function to transition a message to the error database. In this stage, it gets the TPS context from the DTD data. Then it sets the mode in TPS context to `TPS_MODE_RUN`, and runs TPS run mode code. After this completes, the message is written to the error database or goes somewhere else depending on the disposition state returned from TPS.

- **TPS_MODE_SHUTDOWN**

When the thread is shut down, it calls the DBI shutdown function to release resources that are related to the error and recovery database. In this stage, it runs the TPS shutdown mode code. Then, it frees the TPS context and destroys the Xlate thread data and protocol thread data to free the resource.

Note: `TPS_MODE_TIME` is not supported.

Error database TPS message flow

When the process transitions a message to the error database, it goes through these steps:

- 1 Updates the monitorD statistic information that is based on the EDB state of the message, then goes to the next step.
- 2 Updates the recovery database state of the error message. If this fails, then it returns with an error; otherwise, it goes to the next step.

- 3 If message tracing is enabled, then it writes the tracing information for the error message. If this fails, then it returns with an error; otherwise, it goes to the next step.
- 4 If an error database TPS is configured, then it calls the TPS procedure stack and passes the error message and error context for processing. After TPS processing, it does the callback function for disposition. When the disposition is continue, it goes to next step. Or, it goes to Step 1 when the disposition is error, or ends/drops the message when another disposition is found.
- 5 Writes the error message to the error database. If this fails, then it returns with an error; otherwise, it goes to the next step.
- 6 Writes the error message context to the error database. If this fails, then it returns with an error; otherwise, it goes to the next step.
- 7 Removes the message from the recovery database.

Example: Finding and resending a message in ErrorDB

User 1 needs to search the Error Database for a specific message, edit it, and resend it.

To do this, User 1 launches the Error DB IDE tool and performs a regular expression search for “12345.”

One or more transactions are returned from the search. User 1 selects a transaction and edits it to change the user ID from “12345” to “54321.” The edits are temporary and not made to the stored data.

User 1 then resends the transaction.

Network Monitor

The Network Monitor controls and monitors the connections on the network. Network Monitor automatically generates a network control and monitoring panel from a given network configuration file, created through Network Configurator. The default network configuration file for any site is `$HCISITEDIR/NetConfig`.

The Network Monitor cannot access the engine unless the host server is running.

In no security or basic security and running without an audit server, the IDE can be opened in local mode without a running host server. In advanced security, the IDE cannot run in local mode without a running host server.

In no security or basic security and running without an audit server, the IDE can be opened in local mode without a running host server

To start the host server, change to the system root directory and run `hciss -s h`.

A maximum of one (1) Network Configurator can run on the IDE at any given time.

Use the Network Monitor to:

- Start, restart and stop processes and threads.
- Show status information about processes and threads.
- Hold and release outbound messages. See Release options.
- Show graphics showing the activity of processes and threads.
- Forward messages to a different thread.

- View event scheduling information.

The Network Monitor makes use of several floating dialog boxes in the IDE:

- When selected, the Process Control frame is shown along the right side of the tool.
- When selected, the Command and Engine Output frame is shown on the bottom of the tool.
- When Network Monitor is not the active tool, other dialog boxes/frames are not shown. When the Network Monitor becomes the active tool, the other dialog boxes/frames display.
- The floating frames (Process Control and Command and Engine Output) can be set anywhere on the screen. When you re-open Network Monitor, these screens retain their last set location.

Site daemons

To perform its tasks, the Network Monitor depends on the Monitor Daemon, which passes information and instructions between the Network Monitor and the engine.

Lock Manager (UNIX)

In UNIX systems, the Network Monitor also depends on the Lock Manager. The Lock Manager prevents files from being accessed by more than one user at a time. This protects the files from corruption.

The Lock Manager must also be started before you start the Network Monitor.

File lock

When Network Monitor is open and another user attempts to access Network Monitor, a message opens stating the Network Monitor view file is locked.

Network Monitor Properties dialog box

Selecting **Options > Network Monitor Options** opens the **Network Monitor Properties** dialog box.

General tab

This table shows the configuration options on the **General** tab:

Option	Description
Statistics Updates	Specifies the thread polling interval. Click the arrow to select from the list.
Monitor Daemon Polling	Specifies the Monitor Daemon polling interval. Click the arrow to select from the list.
Lock Manager Polling	This option is only available for UNIX hosts. The polling intervals determine how often Network Monitor checks threads and site daemons for status.

Option	Description
Automatically display new alerts	<p>Automatically shows new alerts. This is the default. Clear this to disable this feature. If this is cleared, then Automatically select new alerts is also cleared. In this configuration, you always have some notification that a new alert has arrived. Alerts arrive through the Alerts dialog box or the Network Monitor status bar icon.</p>
Automatically select new alerts	<p>Automatically selects new alerts so that the details are visible. This is the default.</p> <p>Clear this to disable this feature. If this is cleared, then when a new alert arrives, an icon displays on the Network Monitor status bar, indicating that there is an unread alert.</p> <p>Click the icon to open the Alerts dialog box. If this dialog box is already showing, then it is brought to the front. The alert icon remains on the status bar until all alerts have been read.</p> <p>If the Alerts dialog box is minimized, then Network Monitor cannot restore it when a new alert arrives.</p>
Only load the last 10 lines of a process log	<p>Keeps from having to resize the Watch Log File for process dialog box (Control > Full).</p> <p>When this is selected, the dialog box loads only the last 10 lines and remembers the last opened size.</p> <p>The default is cleared. These functionalities also apply to the Message Archive Log Viewer.</p>

Engine Output tab

Use this tab to specify which messages are shown or ignored.

See [Configuring the Engine Output tab](#) on page 184.

Depth Colors tab

Use this tab to determine and select various message queue depth colors on the Network Monitor Grid tab. Queue depth colors represent the number of messages queued for the connection, but not yet delivered.

- View tab colors apply to the individual route lines.
- Grid tab colors apply to the thread.

The depth color is automatically applied to the route lines on the View tab.

The Depth pane lists the various queue depths to which colors have been assigned.

This table shows the configuration options for depth colors:

Option	Description
Add Depth	Click to add another depth to the listings on the Depth pane. For example, you could add a listing for "equal to or less than 6."
Delete Depth	Click to delete the highlighted listing in the Depth pane.
Color cell	The color that is shown is the current color for the selected queue depth. Click the color sample to open the Choose Thread Color dialog box, where you can select another color swatch or specify the particular color. Reset reverts to the default settings.

Protocol Colors tab

The protocol color is shown in a rectangular bar under the protocol status message and is colored according to the protocol status.

This protocol color is updated when any protocol status is changed.

By default, every protocol color is distinct from the default depth colors, which are red, green, and yellow. All protocols have the same status color settings.

Grid View tab

This table shows the options for determining whether to show the thread name and protocol status on the Grid tab.

Option	Description
Show thread name	Select/clear this to show/hide the thread name on the Grid tab.
Show protocol status	Select/clear this to show/hide the protocol status on the Grid tab. The Preview pane shows what the grid display settings display on the Grid tab.

Configuring the Engine Output tab

Use this tab to specify which messages are shown or ignored.

To add a new engine output filter:

- 1 In the **Regular Expression** text box, specify the regular expression or string contained in the output message to filter.

- 2 Select the **Action** for filtering the output (**Display** or **Ignore**).
- 3 (UNIX) To select different colors for displayed output, specify the name or hexadecimal value of any valid UNIX color in the **Foreground** text box.
- 4 Specify a comment in the **Comments** text box describing the filter.
- 5 Select the **Number of lines displayed per process** using the up or down arrows.
- 6 Select the **EO Update Interval** in seconds using the up or down arrows.
- 7 Click **OK** to add the new filter.

Setting the protocol status color

The protocol color is set on the **Protocol Colors** tab of the **Network Monitor Properties** dialog box.

- 1 Select **Display protocol status color** to configure the protocol color. This is displayed on the protocol color bar of the thread icon. This check box is not selected by default, indicating no protocol status is shown on the thread icon.
- 2 Select the protocol from the **Protocol** list, which lists all possible protocols.
- 3 The table in the remainder of the dialog box shows the selected colors for individual protocol statuses. This table has two columns: Protocol Status and Color. Click the column header to sort the columns.
- 4 Click the button to open the **Color Chooser** dialog box, from which you can select the color for the protocol status on that same row. The Color cell shows the selected color as its background color. You can set another color for each individual protocol status.
- 5 Click **OK** to commit the new settings. These settings are immediately applied to the thread icons in the View tab.

The protocol status color settings are user-specified client settings, and are saved in `client.ini`.

For example:

```
[user_<name>_100.0.0.0_5.6_chapard_netmonitor]
protocolcolors=true
protocolcolors_file=-8355712 -16777216 -16777216 -65536 -16731648 -10040065 -10040065 -16777038
-20561
protocolcolors_tcpip=-16737895 -16777216 -16777216 -16777216 -16731648 -10040065 -10040065
-16777038 -20561
```

The value for the `protocolcolors` key indicates the **Display protocol status color** status:

- `true` means display.
- `false` means do not display.

After `protocolcolors` are the protocol status color settings for the individual protocols.

- The protocol name is in the key. This must be lowercase.
- The protocol status color settings for individual protocols following the fixed prefix `protocolcolors_`.
- The color setting is a number that represents the RGB color value in the default RGB integer. Bits 24-31 are alpha, 16-23 are red, 8-15 are green, and 0-7 are blue.
- The delimiter among colors are an empty space.
- The color values are in a fixed sequence for individual protocol status.
- If a protocol is not found here and `protocolcolors` is `true`, then the default color is used.

- If no `protocolcolors` setting is found, then no protocol status color is shown, which works the same as having `protocolcolors` be `false`.

Alerts

Click **View > Alerts** to open the **Alerts** dialog box, where you can view alert messages that have been generated but not yet deleted. Select an alert on the alert list, and its details display in the text boxes.

This table shows the alert fields:

Field	Description
First Received	Shows the date and time the alert was first received. This might be the same as Last Receive.
Last Receive	Shows the date and time the alert was most recently received.
Duplicates	Shows the number of duplicates of the alert that have been received. If no duplicates have happened, then this value is 0.
Host	Shows the host where the alert was generated.
Root	Shows the root for which the alert was generated.
Site	Shows the site for which the alert was generated.
Alert text box	Shows the text of the alert message.
Alert list	Determines the contents of the above text boxes. This list provides a summary listing of all the alerts that have been received because Network Monitor was started. By default, the alert list is sorted by date last received. Alerts which have yet to be reviewed or which have a newly-arrived duplicate display in bold.
Remove Selected	Removes the currently selected alert from the alert list. This is available only when at least one alert is selected.
Save Selected	Saves the selected alert to a text file located on the local machine. This is available only when at least one alert is selected.
Status bar	Indicates the current total number of alerts and the number of unread alerts, if any.

Alert list

This is a list of all the available alert messages. Initially, when the dialog box is first opened, the top message is automatically selected. Another message or multiple messages can also be selected.

Use the buttons along the bottom of the dialog box to remove (delete) the selection or to save the selection to a text file.

If a message listing is in bold type, then it indicates the message has not been read or a duplicate of that message has arrived.

By default, the list is sorted in reverse chronological order, last message first. The listing can be sorted differently by clicking a column heading. Change the order from ascending to descending by clicking the same column heading that currently defines the sort order.

If multiple messages are selected, then the top pane keeps the information from the previously selected single message. If that message is deselected, then the top pane shows information about the first (top) selection.

Alert notification

By default, when Network Monitor receives a notification of a new alert from the Monitor Daemon, the **Alerts** dialog box automatically opens. The new alert is selected so that its details are visible.

You can change this behavior by selecting the appropriate check box on the **General** tab of the **Network Monitor Properties** dialog box.

To do this, select **Options > Network Monitor Options** to open the **Network Monitor Properties** dialog box.

This table shows the alert notification options:

Option	Description
Automatically Display New Alerts	<p>Select this to automatically show new alerts through the Alerts dialog box. This is the default.</p> <p>Clear this to disable this feature. If this is cleared, then Automatically Select New Alerts is also cleared. In this way, there is always some notification of a new alert, through the Alerts dialog box or the Network Monitor status bar icon.</p>
Automatically Select New Alerts	<p>Select this to automatically select new alerts so that the details are visible. This is the default.</p> <p>Clear this to disable this feature. If this is cleared, then when a new alert arrives, an icon displays on the Network Monitor status bar, indicating that there is an unread alert.</p> <p>Click the icon to open the Alerts dialog box. If this dialog box is already showing, then it is brought to the front. The alert icon remains on the status bar until all alerts have been read.</p>

If the **Alerts** dialog box is minimized, then Network Monitor cannot restore it when a new alert arrives.

Show thread bitmap

This shows the selected thread bitmap in the view. By default, this option is cleared and the Network Monitor view does not show the thread bitmap.

This is configured from the NetConfig/NetMonitor tab of the **Client Preferences** dialog box (**Options > Client Preferences**).

This setting is saved in the view file and can be loaded again. The Network Configurator ignores the **Show thread bitmap** setting, even though the Network Configurator and Network Monitor share the same view file.

The thread bitmap is shown on the top of the thread graphic and the thread status label is on the bottom.

View list panel

The View List panel lists all the views for the selected site, and offers access to the view control menus.

View tab

The View tab is the default tab when the Network Monitor first opens. It shows a visual map of selected threads and provides complete control over them. View tab colors apply to the individual route lines.

The depth color is automatically applied to the route lines on the View tab.

For example, if the queue depth is two each of connections and routes, then this is four in the total per-thread queue. These can be another color.

List tab

The List tab shows a table that lists all the threads in the current view, with information about each thread.

The entry for each thread includes:

- Thread name.
- Thread status. Icons provide a quick indication of thread status.
- Volume. This is the number of messages processed by the thread during the current session.
- Depth. This is the number of messages in the thread's message queue.

Grid tab

The Grid view shows the thread name and status, protocol status, depth number and color bars that indicate the protocol status (optional) and depth.

The current depth color is based on a summary of the thread depth and route depth. The thread depth number is the depth of the thread. The route depth number is the summary of the route depth from the target thread.

In a cell, the tip box shows the detail depth information as follows:

- Thread: thread_name

- Thread Depth: thread_depth_number
- Route Depth: route_depth_number

The thread depth and route depth are displayed separately.

In each cell, from left to right, are shown the thread status icon. This is followed by the thread name, protocol status, with optional protocol status color underline, and depth number. The color of the line under the depth number represents the depth color setting.

Thread names that are too long are truncated.

The more threads there are, the more columns there are, with a maximum of eight columns. In these cases, the thread name could be truncated.

Double-clicking in a grid cell opens the **Status Report** dialog box for that thread.

Grid cell menu

Right-click a grid cell to open a menu of options.

This table shows the available options:

Option	Description
Control	Options include Start , Stop , Restart , Hold , Release , and Full . When restarting a thread, the GUI can still be operated. The command line output is appended to the Command and Engine Output pane, where you can see the results.
Hold	Holds outbound messages in the outbound queue.
Release	Releases any messages currently being held.
Full	Opens the Thread Controls dialog box.
Sort	The grid cell is sorted from left to right and from top to bottom. By default, the grid cells are sorted by thread name in ascending order. If you select By Thread Name again, the cells sort in descending order. The same behavior applies to By Depth and By Protocol Status .
Metrics	Opens the Engine Monitor dialog box.
Status	Opens the Status Report dialog box.
SMAT	This has the Open Inbound and Open Outbound options.
Schedule	View opens the Schedule dialog box, where you can Enable , Disable , Enable All , or Disable All . Run Now runs all scheduled timer events.

Option	Description
Notes	Opens the Notes dialog box.

Process Monitor function

By default, the Process Monitor panel opens along the right side of the Network Monitor. The Process Monitor panel lists the process indicators for all processes included in the current view. These indicators provide direct control over the operation of the processes.

Processes are sorted by process name in ascending order. This pane is vertically re-sizable. If the name is too long for the pane width, then a horizontal scroll bar displays.

To open a context menu of process management commands, right-click a process indicator.

This table shows the options on the context menu for process management:

Option	Description
Control	Opens the process control context menu, where you can Start , Stop , or Restart the selected process. Select Full to open a dialog box of options. All Processes opens a context menu for starting/stopping/restarting all processes at one time.
Metrics	Opens the Engine Monitor dialog box. In the dialog box, you can review performance statistics for the session, and view graphs that show the performance in specific areas over time.
Conn Status	Opens the Status Report dialog box for the selected process.
Status	Opens the Process Status Summary dialog box.
Watch Output	Shows process output as it happens.
Browse Output	Opens the logfile of process output.
Startup Log	Opens the Startup Log for Process dialog box.
Notes	Notes can be viewed in plain text or HTML. Notes are written on the Network Configurator's Process Configuration dialog box.

Starting, restarting, and stopping multiple processes

A menu opens when you right-click in the Process Control pane, but not on any process button. You can start, restart, or shut down multiple processes.

Selecting **Start**, **Stop** or **Restart** opens the **Select Processes to Start/Stop/Restart** dialog box. This is a list of processes. The dialog box title depends on the menu selection.

When you select **Restart**, this dialog box lists only the running processes.

Multiple processes can be selected from the list. After you click **OK**, the IDE starts/stops/restarts the selected processes. The order to start/stop/restart the processes is the same as the order of the selected processes on the list. The command line output is appended to the Command and Engine Output pane, where you can see the results.

You can also right-click any of the processes to access the **Control > All Processes** menu, which opens the same **Select Processes to Start/Stop/Restart** dialog box.

- If no process is running, then only **Start** is available.
- If all processes are running, then **Stop** and **Restart** are available.
- If some processes are running and some are dead, then **Start**, **Stop**, and **Restart** are all available.

Process Controls dialog box

The **Process Controls** dialog box offers full control of all the threads in a process.

- 1 Right-click the process name on the Process Monitor panel. This opens the context menu for process management.
- 2 Select **Control** to open the process control context menu.
- 3 Click **Full**. This opens the **Process Controls** dialog box for the selected process.
See [Process control options](#) on page 191.

Process control options

This table shows the **Process Controls** dialog box options:

Option	Description
start	Initiates the process by starting all the threads in the selected process.
stop	Shuts down the process by stopping all the threads in the selected process.
start with hold	Starts the threads in the process with the outbound data on hold.
start with sticky hold	Starts the threads in the process with the last hold status.

Option	Description
restart	Restarts the process. When restarting a process, the GUI can still be operated. The command line output is appended to the Command and Engine Output pane, where you can see the results. When restarting a process, the thread status icon on the GUI changes from green to red to green again.
hold	Places a hold on all outbound data queues for all protocol threads in the process.
release	Releases held messages to continue through processing for all protocol threads in the process.
hold reply	Holds transmission of all thread reply messages of the selected process until released.
release reply	Releases hold on all or a specified number of reply messages.
reload	<p>Reloads an updated Tcl procedure. Clicking List on the Reload Procedure dialog box opens a dialog box containing Tcl procedures from which to select.</p> <p>Select/clear Only display procs at local site to hide/show global/master procs to make the procedure list smaller.</p>
purge caches	Removes the configuration caches. Use this option after making a change to an existing configuration. This is not designed to be used for the creation of new threads, processes, routes, and so on.
watch output	Shows engine output as it happens.
browse output	Opens the log file of engine output.
engine log configuration	Opens the Engine Log Configuration dialog box. Click List to select the Engine Output Alias. This applies an Engine Output (EO) alias, configured in the Engine Output Configurator, to the threads in the process.
reindex	Loads Tcl procedures that were created when the engine began running.
cycle output	Causes the current engine log (.log) and error (.err) files to close and be renamed with an .old extension, and then opens new logs.
status	Opens the Status Report dialog box, which provides statistical information on the threads within the process.

Option	Description
reset statistics	Resets statistics and counts for a site's process or thread. This calls the <code>hcimsiutil</code> engine command. Before using this option, there is the potential to produce unreliable counts. The memory is not corrupted when this option is used. The engine and Network Monitor both lock the shared memory for any write/clear operation.

Managing the view

The **View** tab is a graphic representation of all the threads in the currently selected view. If any set of these threads share a part of the same route, then the threads are shown linked together.

Thread icons

Each thread is represented by a thread icon.

A blue outline around the icon indicates the focus.

On a destination site, if there is any route to a public thread, an icon is shown on the upper-right of the public thread. This indicates the referenced public thread. The referenced indicator displays only after synchronization is completed from the source site.

Thread connection status

- Dead: The thread is not running. It has stopped or been shut down.
- Initializing: The thread has been started and the engine is preparing to run it.
- Opening: The thread is attempting to establish a connection.
- Up: The thread is connected and running.
- Timer: An event is scheduled for that thread.
- Down: The thread is running, but is not attempting to establish a connection.
- Closing: The thread is in the process of shutting down the connection.
- Error: The thread connection has failed because of a problem.
- INEOF: A file-based connection has reached the end of the input file.

Thread name

The thread name uniquely identifies the thread.

Protocol state

The protocol state color is shown in a rectangular bar under the protocol state message and is colored according to the protocol state.

This protocol state color is updated when any protocol state is changed.

By default, every protocol state color is distinct from the default depth color (red, green, and yellow). All protocols have the same state color settings.

The protocol state color is set on the Protocol Colors tab of the **Network Monitor Properties** dialog box.

Route lines

If two or more of the threads share a part of the same route, then the threads are shown linked by an arrow.

- The arrow shows the direction of message flow from thread to thread.
- The arrow color represents queue depth.

The colors can also be customized to your site through the **Choose Thread Color** dialog box. This dialog box is accessed by clicking **Change** on the Depth Colors tab of the **Network Monitor Properties** dialog box.

- Route lines stay visible whether the route line is clicked.

To find out more about a route, right-click the route line, and then click **Transaction Summary** to open the **Transaction Summary** dialog box, which includes:

- The number of messages sent across the route during the current session.
- The number of messages waiting in queues.
- The name of the translation thread.
- The type of translation.
- The file that contains the translation specifications.

Click **Refresh** to refresh the run-time route summary.

Depth colors are also applied to reply routes, for example, the color of any reply route is updated based on the reply queue depth. Highlight the route path and reply route path by moving the cursor over the path line.

Updating views

Use Network Configurator to create new threads or to update existing properties.

When a thread is updated or added to an existing process, that thread is automatically added to the corresponding process view in Network Configurator.

When a thread is updated or added to an existing group, it is also automatically added to the group view in Network Configurator.

In Network Monitor, you must refresh the view to see the change (**File > Refresh**).

Note: Built-in views, including the all_threads view, process view, and group view, cannot be updated or edited.

Creating a new view

Use the Context Menu for View Management to create and manage views. You can create a view for a set of threads to be controlled with one start command.

- 1 Right-click in the View List pane. This opens the Context Menu for View Management.
- 2 Click **Add New View**. This opens the **Create View** dialog box.
- 3 Specify a view name in the **View Name** text box.
- 4 Then, select the threads and views to be included in the new view.

Editing an existing view

- 1 Right-click an existing view on the View List pane. This opens the Context Menu for View Management.
- 2 Then, click **Edit View**. This opens the **Create View** dialog box for the selected view.
- 3 Edit as necessary.

View panel

A view panel is located on the left side of the Network Monitor. This panel is similar to the view panel in the Network Configurator. The view panel is a horizontally resizable panel in Network Configurator and Network Monitor.

The view panel lists all views in the current `mvw` file in sorted order. These views can only be single-selected. The selected view is shown in the layout panel. By default, the `all_threads` view is automatically selected.

If any view is changed without being saved, then an asterisks (*) is appended to the view file name in the title.

There are two types of views in the view panel:

- The built-in views includes the `all_threads` view, process views and group views. The `all_threads` view contains all threads in the Network Configurator. The IDE creates a process view for each process. The process view name is named with the syntax, `process_process name`.
The IDE also creates a group view for every group and is named with the syntax `group_group name`. All default created views are refreshed based on the NetConfig file when Network Configurator/Network Monitor is opened.
NetConfig can be changed separately from the view file. The views in the old view file might not be consistent with the new NetConfig file, especially for default created views. The refresh is performed to makes sure the default created views are up-to-date.
- User-created views are used to view threads/processes/groups, and can contain the sub-view which references the other views.
If a thread is created in a group or process view in Network Configurator, then by default the group/process of the newly created thread is the group/process that is currently selected.
For the `all_threads` view and user created views in the Network Configurator, the default process name of a new thread is the site name. The default group name is blank.
You can change the process/group of a thread in a selected view in Network Configurator. If the new process/group does not belong to the selected view, then there is a warning message after you click **OK** to commit the change:
The new process/group of the thread `thread_name` does not belong to current view. The `thread_name` will be removed from this view.

After dismissing the warning message, the changed thread is removed from the current view. It is added to the view to which it belongs.

Managing threads in a view

The Network Monitor offers many ways to manage threads. A straightforward approach is to select a view in the Network Monitor. Then use the context menu on the View List pane to manage all the threads in the view together. You can also use the context menu for a thread icon on the **View** tab to manage an individual thread.

The View List pane lists all the views that have been created for the current site. Each view is a visual map of a selected set of threads. Threads that share a part of the same route are shown linked to each other.

The first (default) view in the list is `all_threads`. This view shows all the threads in the current site. It is created automatically and cannot be removed.

Views are also created automatically for every process and every group that has been configured in the Network Configurator. These views are always named *process_name* or *group_name*.

Starting, stopping, or restarting an individual thread

- 1 Select the view that contains the thread to start, stop, or restart by clicking the view name in the View List pane.
- 2 Select the thread to start, stop, or restart. To do this, right-click the appropriate thread icon on the View tab. This opens the Context Menu for Thread Management.
- 3 Select **Control** to open the thread control context menu.
- 4 Click **Start** to start the thread **Stop** to stop the thread, or **Restart** to restart the thread.

Starting, stopping, or restarting all threads in a view

- 1 Select the view that contains the threads to start, stop, or restart by clicking the view name in the View List pane and select `all_threads`.
- 2 Right-click anywhere in the View List pane to open the context menu for view management.
- 3 Select **Control** to open the view control context menu.
- 4 Click **Start** to start all threads, **Stop** to stop all threads, or **Restart** to restart all threads.

View Controls dialog box

The **View Controls** dialog box offers full control of all the threads in a view.

- 1 In the Network Monitor, click a view in the view list.
- 2 Right-click anywhere in the View List pane to open the context menu for view management.
- 3 Select **Control** to open the View Control Context menu.

- 4 Click **Full**. This opens the **View Controls** dialog box.

View control options

This table shows the options on the **View Controls** dialog box:

Option	Description
start	Starts all the threads in the selected view.
stop	Stops all the threads in the selected view.
start with hold	Starts the threads in the view with the outbound data on hold.
start with sticky hold	Starts the threads in the view with the last hold status.
restart	Restarts all the threads in the selected view.
hold	Holds outbound messages in the outbound queue.
release	Releases any messages currently being held.
hold reply	Holds transmission of all thread reply messages for the selected view until released.
release reply	Releases hold on all or a specified number of reply messages.
reload	Adds any new Tcl procedures.
purge caches	Removes the configuration caches. This action clears all record and lookup table configurations. Use this option after making a change to an existing configuration.
engine log configuration	Opens the Engine Log Configuration dialog box. Click List to select the Engine Output Alias. This applies an Engine Output (EO) alias, configured in the Engine Output Configurator, to the threads in the view.
reindex	Loads Tcl procedures that are created when the engine began running.
status	Opens the Status Report dialog box.

Option	Description
reset statistics	Resets statistics and counts for a site's process or thread. This calls the <code>hcimsiutil</code> engine command. Before using this option, there is the potential to produce unreliable counts. The memory is not corrupted when this option is used. The engine and Network Monitor both lock the shared memory for any write or clear.

View control context menu

Use this to access view-specific commands for the selected view.

- 1 On the Network Monitor, click a view in the view list.
- 2 Right-click anywhere on the View List pane to open the context menu for view management.
- 3 Select **Control** to open the view control context menu.
Start starts the threads included in the selected view.
Stop stops the threads included in the selected view.
Restart restarts the threads included in the selected view. When restarting a view, the GUI can still be operated. The command line output is appended to the Command and Engine Output pane, where you can see the results.
Full opens the **View Controls** dialog box.

Context menu for view management

To open a context menu of view management commands, select a view, right-click anywhere on the View List pane. This opens the context menu for view management.

This table shows the available options:

Option	Description
Control	Opens the view control context menu. In this menu, you can Start , Stop , or Restart the threads that are shown in the view. You can also select Full to open a dialog box of options.
Metrics	Opens the Engine Monitor dialog box. In this dialog box, you can review performance statistics for the session. You can also view graphs that show the performance in specific areas over time.
Status	Opens the Status Report dialog box.

Option	Description
Edit View	Opens the Create View dialog box with the specifications for the selected view. Use this dialog box to add or delete threads or other views.
Delete View	Deletes the selected view, after prompting you to confirm your intention.
Filter View	<p>Opens the Thread Filter Settings dialog box. Use this dialog box to control which threads display in the current view.</p> <p>Viewing criteria are ANDed together. For example, if all thread names are listed, and a single group name is listed, only those threads within that group are shown.</p> <p>You can define the view limits to a set of specific processes, group names, or specific threads.</p> <p>The Filter View accepts names that are partial or case-insensitive.</p> <p>Specify the partial name of an object for which you are looking to find that object.</p> <p>This search is case-insensitive. For example, if the threads are named <code>phy_beta</code> and <code>phy_kappa</code>, specifying <code>phy</code> locates both names.</p>
Add New View	Opens the Create View dialog box with no view specifications. Use this dialog box to create a new view that contains the selected threads or views.
Import View	<p>Opens a file browser, from which you can select a view to import. The filter settings in the <code>.view</code> file are based on the current NetConfig file during importing.</p> <p>All threads in the view file and in the current NetConfig file are imported into a new single view configuration.</p>

Managing threads

To open a context menu of thread management commands, right-click anywhere on a thread icon. This opens the context menu for thread management.

This table shows the options for managing threads:

Option	Description
Control	Opens the thread control context menu with a list of options. You can also select Full to open a dialog box of additional options.
Metrics	Opens the Engine Monitor dialog box, where you can review performance statistics for the session, and view graphs that show the performance in specific areas over time.
Status	Opens the Status Report dialog box.
SMAT	<p>Open Inbound and Open Outbound open a SMAT Database tab to the *_in.smatdb or *_out.smatdb file. The file type opened depends on which SMAT driver is being used. The SMAT driver can be changed by selecting/clearing Save into Database on the SMAT tab of the Site Preferences dialog box.</p> <p>If it is selected, then the SMAT database file opens if it has been configured; otherwise, the SMAT plain file is opened.</p>
Schedule	<p>This is enabled when Advanced Scheduling is configured for the thread.</p> <p>View opens the Schedule <thread> dialog box.</p> <p>Run Now runs the schedule for threads with a timer. If a thread has advanced scheduling that is configured through the Network Monitor, then you can also use the <code>hcicmd runnow_schedule</code> command to run the scheduled task. For example, <code>hcicmd -p <proc_name> -c "<conn_name> runnow_schedule"</code>.</p>
Notes	Opens the Notes dialog box for that thread. This option is available only when there are notes, indicated by the icon on the thread.
Create View	Opens the Create View dialog box, where you name a new view. Ctrl-click as many threads as are required for that view.

Viewing scheduled events

Events are scheduled for Fileset Local, Fileset FTP, HTTP Client, or UPoC.

Use the Network Configurator to schedule events.

Use the Network Monitor to enable or disable events.

When a schedule is set, the timer symbol displays on the thread icon. The clock icon also displays on the Grid pane. Right-click the thread icon to open the context menu with **View Schedule** enabled.

Select **View Schedule** to view that thread's schedule.

Use the **Schedule** dialog box to disable or enable that thread's schedule.

Thread Control context menu

Use the thread control context menu to access thread-specific commands for the selected thread. Thread Control context menu

- 1 In the Network Monitor, click a view in the view list.
- 2 Right-click a thread icon on the **View** tab to open the context menu for thread management.
- 3 Select **Control** to open the thread control context menu.

Thread control context menu options

This table shows the thread control context menu options:

Option	Description
Start	Starts the selected thread. If necessary, then the process containing the thread is started.
Stop	Stops the selected thread. The process containing the thread is not stopped.
Restart	Restarts the selected thread. If necessary, then the process containing the thread is restarted.
Hold	Holds outbound messages in the outbound queue.
Hold Reply	Holds transmission of all reply messages for the selected thread until released.
Release	Releases any messages currently being held in the selected thread.
Release Reply	Releases hold on all or a specified number of reply messages.
Full	Opens the Thread Controls dialog box.

Thread Controls dialog box

Use the thread control context menu to access thread-specific commands for the selected thread.

- 1 In the Network Monitor, click a view in the view list.

- 2 Right-click a thread icon on the **View** tab to open the context menu for thread management.
- 3 Select **Control** to open the thread control context menu.
- 1 In the Network Monitor, click a view in the view list.
- 2 Right-click a thread icon on the **View** tab to open the context menu for thread management.
- 3 Select **Control** to open the thread control context menu.
- 4 Select **Control** to open the thread control context menu.
- 5 Click **Full**. This opens the **Thread Controls** dialog box for the selected thread.

Thread Controls dialog box options

This table shows the thread control options:

Option	Description
start	Starts the selected thread.
stop	Stops the selected thread.
start with hold	Starts the thread with the outbound data on hold.
start with sticky hold	Starts the thread with the last hold status.
restart	Restarts the selected thread. When restarting a thread, the GUI can still be operated. The command line output is appended to the Command and Engine Output pane, where you can see the results. When restarting a thread, the thread status icon on the GUI changes from green to red to green again.
hold	Holds outbound messages in the outbound queue.
release	Releases any messages currently being held.
hold reply	Holds transmission of all reply messages for the selected thread until released.
release reply	Releases hold on all or a specified number of reply messages.
start save	Starts the saved message file.
stop save	Stops the saved message file.

Option	Description
cycle save	<p>Causes the current saved message (SMAT) file to close and be renamed with an <code>.old</code> extension, and then opens a new saved message file. Each thread can have at most two SMAT files, one for inbound data and one for outbound data. The names and locations of these SMAT files are user-definable.</p> <p>The engine automatically turns off saved messaging when disk space reaches a certain threshold (90%). It turns on again when the percentage used drops below the threshold. This does not affect the actual messages successfully sent to the receiving system. Saved messages are copies only of messages sent when saved messaging is active.</p>
resend	Opens the Resend Datafile dialog box. Use this dialog box to insert a set of messages from a stored file into a running engine.
reload	Adds any new Tcl procedures.
purge caches	Removes the configuration caches. Use this option after making a change to an existing configuration.
start forward	Starts forwarding data messages to a configured destination.
stop forward	Stops forwarding data messages.
start reply forward	Starts forwarding reply messages to a configured destination.
stop reply forward	Stops forwarding reply messages.
watch output	Shows engine output as it happens.
browse output	Opens the logfile of engine output.
engine log configuration	<p>Opens the Engine Log Configuration dialog box. Click List to select the engine output alias. This applies an engine output (EO) alias, configured in the Engine Output Configurator, to the thread.</p>
reindex	Loads Tcl procedures that were created when the engine began running.
status	Opens the Status Report dialog box.
message archive log	Opens the Message Archive Log Viewer for the archive message logs.

Option	Description
open inbound SMAT	<p>Opens the corresponding <code>*_in.smatdb</code> file configured in the Network Configurator's Properties tab, In-bound pane.</p> <p>The file type opened depends on which SMAT driver is being used. The SMAT driver can be changed by selecting/clearing Save into Database on the SMAT tab of the Site Preferences dialog box.</p> <p>If it is selected, then the SMAT database file is opened if it has been configured; otherwise, the SMAT plain file is opened.</p>
open outbound SMAT	<p>Opens the corresponding <code>*_out.smatdb</code> file configured in the Network Configurator's Properties tab, Outbound pane.</p> <p>The file type opened depends on which SMAT driver is being used. The SMAT driver can be changed by selecting/clearing Save into Database on the SMAT tab of the Site Preferences dialog box.</p> <p>If it is selected, then the SMAT database file is opened if it has been configured; otherwise, the SMAT plain file is opened.</p>
reset statistics	<p>Resets statistics and counts for a site's process or thread. This calls the <code>hcimsiutil</code> engine command.</p> <p>Before using this option, there is the potential to produce unreliable counts. The memory is not corrupted when this option is used. The engine and Network Monitor both lock the shared memory for any write/clear operations.</p>

Message resend

You can use the **Destination Threads** field on the **Resend Data File** dialog box to specify the destination threads for message resends. This is the **Thread Controls** dialog box option.

When specifying the destination thread for the resend, inter-site threads are not supported.

Destination threads are relevant to the source thread and resend **Message Type**:

- **Reply:** Only those threads defined as reply route destinations of the source thread can be specified as the destination threads.
- **Data:** Only those threads defined as route destinations of the source thread can be specified as the destination threads.

List is unavailable in these conditions:

- **Reply** is selected and the selected source thread does not have the reply route destinations.
- **Data** is selected and the selected source thread does not have the route destinations.

You can click **List** to open the **Select Destination Threads** dialog box. In this dialog box, you can select the destination threads from a thread list.

- If **Reply** is selected, then the dialog box lists the available reply route destination threads for the source thread.
- If **Data** is selected, then the dialog box lists the available route destinations threads for the source thread.

If the resend **Context** is **Outbound pre-TPS** or **Outbound post-TPS**, then **Destination Threads** and **List** are unavailable.

The **Encoding** list is used to specify the encoding for resent messages. The user-selected encodings are passed from the list to the resend command line.

For the default value of the list Network Monitor's **Resend Data File** dialog box, the NetConfig thread encoding is used, if there is one. If not, then the default encoding is used.

Releasing messages of a process

Using the release option, you can release one or more messages without removing the hold on the thread. You can also hold and release reply messages.

- 1 Right-click a process in the process or View List pane, and select **Control > Full**. This opens the **Process Controls** dialog box opens.
- 2 Select **Release/Release Reply**. This opens the **Process name Release/Release Reply Options** dialog box.

Selecting **Release all currently held messages** removes the hold on the thread after all the messages in the outbound queue are released.

When using this option, the command line `hccmd -p process_name -c "thread_name_list prls_obd"` is appended to the Command Output pane, if it is open.

Select **Release only the first <n> message(s) and keep the thread on hold** to specify how many messages to release in the outbound queue and still maintain the hold on the thread.

When using this option, the command `hccmd -p process_name -c "thread_name_list prls_obd count_of_messages"` is appended to the Command Output pane, if it is open. After applying this option, the thread is still in a hold status.

Use `prls_obd count_of_messages` to specify the number of messages to release.

When this option is selected the spinner is enabled, accepting digits from 1 to 9999. If the count is zero or missing, then the command releases the hold.

- 3 Click **OK** to apply the option, or **Cancel** to cancel the operation.

Releasing messages of a thread

Using the release option, you can release one or more messages without removing the hold on the thread. You can also hold and release reply messages.

- 1 Right-click a thread on the **View** tab or **Grid** tab. This opens the thread control context menu, where you can select the **Release/Release Reply** option.

Release/Release Reply can also be accessed from the **Thread Controls** dialog box (**Control > Full > Release/Release Reply**).

- 2 Select **release** or **release reply**. This opens the **Thread name Release/Release Reply Options** dialog box.

Selecting **Release all currently held messages** removes the hold on the thread after all the messages in the outbound queue are released.

When using this option, the command line `hcicmd -p <process_name> -c "<thread_name> prls_obd"` is appended to the Command Output pane, if it is open.

Select **Release only the first <n> message(s) and keep the thread on hold** to specify how many messages to release in the outbound queue and still maintain the hold on the thread.

When using this option, the command line `hcicmd -p <process_name> -c "<thread_name> prls_obd <count of messages>"` is appended to the Command Output pane, if it is open. After applying this option, the thread is still in a hold status.

Use `prls_obd count_of_messages` to specify the number of messages to release.

When this option is selected the spinner is enabled, accepting digits from 1 to 9999. If the count is zero or missing, then the command releases the hold.

- 3 Click **OK** to apply the option.

Declaring variables

Translation variables are not protected from variables in other xlates.

You must declare all variables before using them and explicitly set the variable values; otherwise, the variable values are not as expected.

In this example, the scope of variable `z` is in the xlate `tcl interp`, which is in an xlate `
thread`.

Then, in another xlate file that is located in the same xlate thread, you can get the variables value. `test1.xlt` and `test2.xlt` to use the `
`same TCL interpreter, so that the variable in `test1.xlt` is visible in `test2.xlt`.

For example, there are two xlate files, `test1.xlt` and `test2.xlt`. In `test1.xlt`, there is a variable `z`.

In `test2.xlt`, there is no variable to declare. However, you can echo the `z` variable. This variable is not protected. You can use the variable `z` (declared in `test1.xlt`) in `test2.xlt`.

If you use variable `z` in `test2.xlt`, and the value of `z` is not the same as the value in `test1.xlt`, then you must reset it.

- 1 On the **Translation Configurator**, for **Action**, select "COPY".
- 2 On the **Pre Proc** tab, select **Tcl**.

- 3 In the field, add:

```
set z xxxx
puts "xlate1 $z"
set in $xlateInVals
foreach a $in {
  if {$a ne {}} {
    lappend liz $a
  }
}
```

- 4 To `
`same to use the same variable name in `test2.xlt`, you must reset it. To do this, in the field, add:

```
puts "xlate2 -- $z"
set in $xlateInVals
foreach a $in {
  if {$a ne {}} {
    lappend liz $a
  }
}
```

Finding the next messages to be sent out on a thread

When there is a backlog of messages to be sent, it can be useful to know what the the next transaction is going to be.

For example, there could be an issue with a transaction that abruptly ends a process. It would be helpful if you could know the queue order of the next transactions to be sent.

You can find the next messages to be sent out to a thread from the recovery database using filters and sorting with `hcidbdump`.

- 1 On the command line, specify `hcidbdump` with the recovery database:

```
hcidbdump -r
```

If the source thread is `conn_1`, append the `-f conn_1` flag.

- 2 Specify the state. For example, state 11 is: `-s 11`

- 3 Sort by outbound time. For example: `-o o`

This displays the messages to be sent out next from the recovery database. For example:

```
C:\cloverleaf\cis6.1\integrator\639442_orig>hcidbdump -r -s conn_1 -s 11 -o o
```

The output is:

Created	Message Id	s e d	Prio	State	Length	Source	Dest
16:52:14	[0.0.8]	P D N	5120	11	594	conn_1	conn_2
16:52:14	[0.0.10]	P D N	5120	11	582	conn_1	conn_2

16:52:15	[0.0.12]	P D N	5120	11	575	conn_1	conn_2
16:52:16	[0.0.14]	P D N	5120	11	591	conn_1	conn_2

Performance metrics graphs

Each option in the **Graphs** menu of the **Engine Monitor** dialog box opens a graph of another metric. Although each graph measures another aspect, all have the same basic design.

The numbers on the left are ranges for the metrics. They are automatically adjusted to show other ranges for the actual numbers. For example, the graph for bytes per second might range from 4000 to 7000, with a horizontal line at every 500.

The numbers at the bottom are times at which the graph is updated, in hh:mm:ss format. These times depend on the interval selected for updates. The breaks differ slightly because of the small amount of time it takes to process the information and change the display.

Every time the graph is updated, the newest interval displays on the right side and the oldest scrolls to the left side. To freeze the graph, use the scroll bar at the bottom by scrolling to the left. The graph stays wherever you stop the scroll bar.

Reviewing performance

The Network Monitor context menus have a **Metrics** option. Selecting this option opens the **Engine Monitor** dialog box that contains statistical information about the performance of:

- A selected thread.
- A selected group of threads.
- All the threads in a selected process.
- All the threads in a selected view.

When the **Engine Monitor** dialog box is first opened, it shows statistics that are related to the entire time that the threads have been running.

Graphs menu options

Graphs can be shown individually or multiply. These graph options are available:

- **Outbound Processing Latency** opens a graph showing the outbound processing latency. Latency refers to the time taken to process a message.

The metric unit is in seconds.

The yellow graph is the total latency for messages in the process. This time includes inbound processing, translation, outbound processing, and queue time.

The units on the left side of the graph are an aggregate of all messages currently in the engine. For example, if there are 100 messages in outbound queues waiting to be sent, the number is the sum of all their wait times.

- **Msgs/Sec** opens a graph showing the average number of messages that are processed per second.
- **Bytes/Sec** opens a graph showing the average number of bytes that are processed per second.
- **Txns/Sec** opens a graph showing the average number of transactions that are processed per second.

Totals menu option

Show Totals shows the Totals pane in the **Engine Monitor** dialog box. When opening multiple graphs, hide the Totals pane to better see the graphs.

Interval menu options

These set the interval between updates to the information in the **Engine Monitor** dialog box. The available intervals range from 5 seconds, the default, to 5 minutes.

Totals pane

This table shows the metrics for a time period:

Metric	Description
Txn Count	The total number of transactions that are processed.
Txns/Sec	The average number of transactions that are processed per second.
Send Time	The average number of seconds to send a message to its destination. This is an external connection or a translation thread.
Xlt Time	The average number of seconds to translate a message.
Msgs In	The total number of inbound messages that are received.
Bytes In	The total number of bytes that are received for inbound messages.
Msgs Out	The total number of outbound messages that are sent or delivered.
Bytes Out	The total number of bytes that are sent or delivered for outbound messages.
Up	The number of threads that are controlled by this process that are currently running. This statistic is available only when viewing the metrics of a process.
Down	The number of threads that are controlled by this process that are currently not running. This statistic is available only when viewing the metrics of a process.

Metric	Description
State	The current state (up, down, and so on) of the selected thread. This statistic is available only when viewing the metrics of a thread.

These metrics are updated at regular intervals, with the interval between updates that are determined by the selection in the **Interval** menu.

Reviewing the status

The Network Monitor context menus also have a **Status** option, which opens a **Status Report** dialog box.

The status report covers all the threads listed in the title bar of the dialog box, listing a separate report for each thread. By default, the **Status Report** dialog box opens to show the last report in the list.

To update the status report automatically, select **Auto-update**.

This table describes the reported parameters:

Parameter	Description
Process	The process name of the thread.
Connection	The name of the thread (connection) for the current status.
Reported at	The date and time of the status report.
Status	The thread state at the reported time.
Started/Stopped	The date and time that the thread was last started or stopped. This depends on the status.
Msg in	The number of messages that are read by this thread from its protocol connection.
Last Rd	The date and time that the messages were last read by this thread from its protocol connection.
Bytes in	The number of bytes that are read in by this thread from its protocol connection.
Msgs out	The number of messages that are written out by this thread to its protocol connection.
Last wt	The date and time that the messages were last written out by this thread to its protocol connection.
Bytes out	The number of bytes that are written out by this thread to its protocol connection.

Parameter	Description
Proto Info	The number of connected clients for the TCP/IP and PDL TCP/IP protocols.
Proto Err	The date and time of the last protocol error.
Error Msg	The text message of the last protocol error.
Xlated	The number of messages that are translated by this thread.
Pending	The number of pending messages for this thread. For example, messages waiting to be delivered to a destination thread or outbound connection that is down.
Forwarded	The number of messages that are forwarded to an alternate host, after the failure of all transmission retries to the primary destination host.
Failed	The number of messages that were not delivered successfully.
IB Lat	Inbound Latency. The number of cumulative seconds that this thread has taken to process inbound messages. This is recorded after the translation is complete and corresponds to the translation count (XLA TECNT). This records the time period between message reception (for example, recovery, Tcl, IB protocol, and so on) and successful completion of the translation.
OB Lat	Outbound Latency. The number of cumulative seconds that this thread has taken to process and deliver outbound messages. This is recorded after a message is successfully written before writing to SMAT, before SENDOK TPS is processed) and corresponds to messages out (MSGOUT). This records the time period between successful completion of the translation and successful send. If a message never went through translation, for example, it was created in OB Data TPS, then this is the same time it was received/created.

Parameter	Description
Total Lat	<p>Total Latency. The total of inbound and outbound latency in seconds (cumulative). This is recorded after a message is successfully written before writing to SMAT, before SENDOK TPS is processed) and corresponds to messages out (MSGSOOUT). This records the time period between message reception (for example, recovery, Tcl, IB protocol, and so on) and successful send.</p> <p>Dividing these numbers by the number of corresponding messages that were processed yields the average message latency: in other words, the average time each message spent in the thread.</p>
Sent	The number of messages that are sent by this thread.
Recvd	The number of messages that are received by this thread.
pxqd	Pending Xlate Queue Depth. The number of messages (queue depth) that the thread has sent to its destination translation thread that have not yet been processed.
Xlt time	The total number of seconds that are required to translate a message.
T on Q	Time on Queue. The total number of seconds spent waiting on queues (T on Q). This is updated when the message is delivered, so T on Q values exist only for threads which have delivered messages. Compare this value to Total Lat to evaluate the amount of processing time versus the pending time.
Latency	<p>The total number of seconds that the thread spent on the same last message, from receipt to delivery.</p> <p>Subtract "T on Q" from "Tot Lat" to find the processing time.</p>
Thread name	The name of the sending or receiving thread.

Remote commands tool

Remote commands are a means of accessing tools through the command line. This includes:

- System commands

By default, these are located under the site-commands node of the ACL tree.

- Engine commands
- Miscellaneous commands
- MonitorD commands
- Testing commands

Remote commands can run standard system commands remotely without the use of a telnet session.

Remote commands act as an interface from which you run commands as if you were at a command line that is initialized for your current system root and site. After the command is run, you can monitor its output and send responses to queries by the remote process.

In addition to the standard system commands, you can also run your own commands. You can do this as long as they are in the `/integrator/bin` directory or the `/integrator/usercmds` directory.

This tool can also terminate a running process.

When used with security server, you can run only those command tools for which you have permission in the ACL (Access Control List) tree. This is defined using the ACL Role Manager.

Remote commands GUI options

Use these GUI options in selecting and running commands:

- Process menu:
 - **Erase output before each run** determines whether the process output is cleared before a process is run. By default, this option is selected.
 - **Clear output** clears the current output from the **Output** list box.
 - **Save output to file** saves the current output to a text file on the local machine.
 - **End process** ends the current process. If no process is running, then this option is disabled.
- The Command pane is used to define and specify commands to process. The **Command** list box shows the currently selected command.
 - **List Commands** opens the **Select Command** dialog box. Use this dialog box to select a predefined command to run. Commands are divided into sections by functionality, along with an additional section which lists user-defined commands. When a command is selected, the template for running the command is shown in the text box. Double-click a leaf node or click **OK** to copy the template to the **Command** list box. If the template contains fields that are replaced by the user, then those fields are italicized. Click **Refresh** to return the **Select Command** dialog box to its previous setting.
 - **History** opens the **Command History** list box. This lists all successfully run commands, beginning with the most recent. Double-click a command from the list or click **Apply** to place it in the **Command** list box of the Remote Commands tool. Then, click **Execute** to run the command. When a process is running, the controls in this section are disabled.
 - **Execute** runs the current command in the **Command** list box.
 - The default **Output Encoding** is UTF-8. This works with all system command lines. If you add your own command line, then the output might not be UTF-8. In these cases, there is an **Output Encoding** menu, where you can select the encoding that is applied on the output of the command line.
- Use the Process pane to monitor the process output and interact with the process as necessary.

- **Input** specifies a response to an input request from the process. This field is enabled when the process is running, even if no user input is required. All user input must be entered in this field.
- **Send** sends the input response. You can also press **Enter**. The **Input** field is enabled when the process is running, even if no user input is required.
- **End Process** ends the current process. If no process is running, then this option is disabled.
- The output shows the command and any user options. All options must be entered in the **Input** field.
- **Clear Output** clears the current output from the Output list box.
- **Save Output to File** saves the current output to a text file on the local machine.

Wildcard support in Remote Commands

When you enter a wildcard at a UNIX command prompt, it is the shell program, for example, `bash`, that interprets it. An example of a wildcard is `grep exec /opt/cloverleaf/cis20.1/integrator/*/tclprocs/*.tcl`.

The **Remote Commands** tool is not a shell command interpreter. It runs the command. That is why it appears that wildcards are not supported in the **Remote Commands** path argument. The pipes, or redirect in the command lines, are interpreted by the shell. **Remote Commands** does not interpret them.

To use wildcard, pipes, or redirect in the **Remote Commands**, there are two alternative methods:

- [Method 1: Adding a shell interpreter before the command line](#) on page 214
- [Method 2 \(recommended\): Write a shell script and wrap the command](#) on page 214

Method 1: Adding a shell interpreter before the command line

Add a shell interpreter before the command line. For example, `/bin/bash -c "grep exec /opt/cloverleaf/cis20.1/integrator/*/tclprocs/*.tcl"`.

This method requires adding the shell interpreter into the command line allowlist and restarting the Host Server. For example, `/bin/bash`.

This is not the safest method to open the shell interpreter in the allowlist.

Method 2 (recommended): Write a shell script and wrap the command

This method writes a shell script and wraps the command that was run in it. This method is safer and does not open `/bin/shell` in the allowlist.

As an example of wildcards:

- 1 Write a `custom_grep.sh` script. For example:

```
user_arg=$1
grep $user_arg
```

- 2 Add `custom_grep.sh` to the allowlist and restart the Host Server.

- 3 In the **Remote Commands** tool, run this command:

```
/home/user/custom_grep.sh exec /opt/cloverleaf/cis20.1/*/tclprocs/*.tcl
```

As a pipes example, write a piping.sh:

```
user_arg1=$1
user_arg2=$2
/bin/bash -c "$1 | $2"
```

- 4 Add it to the allowlist.
5 Restart the Host Server
6 Run it in **Remote Commands**:

```
/opt/cis20.1/integrator/usercmds/piping.sh "ls -l $HCIR00T" "grep hello"
```

Creating and using user-defined commands

When using remote commands, you can select from a list of predefined system commands. You can also select from your own user-defined commands, as long as they are in the `/hci/server` or `/hci/usercmds` directory.

You can also right-click under a user-defined command folder to create sub-categories, rename a category, or delete a category.

The user-defined category structure is kept in a file named `user_cmds.dat` and saved to the `hci/server` folder, which is shared by every client. The actual user-command files are put under the `hci/usercmds` folder.

Use these guidelines when writing commands using Remote Commands:

- User commands must not make any assumptions about their working directory.
They must rely on command-line arguments to specify any directories or files with which to work. The only exception to this is the system directories.
Because the commands are run in a system shell, expect the standard environment settings to be present in the process environment. Standard environment settings include `HCIR00T` and `HCISITE`.
- Write user commands so that they automatically flush the standard output buffer when it is written to. Otherwise, the Remote Commands tool might not display the output in a timely fashion.

Under certain host platforms, you can run some of the standard shell commands, for example, `ls`, `cp`. This is not guaranteed. Do not rely on this functionality to accomplish your work. For full remote shell support, use telnet or a similar remote shell environment.

To use a user-defined command:

- 1 Create a command and copy it into `hci\usercmds`.
- 2 Launch the IDE and open the Remote Command tool.
- 3 Click **List Commands**.
- 4 Expand the User-Defined Commands node to verify that the command is there.

If the **Select Command** dialog box is open before you copy the command to `hci\usercmds`, then click **Refresh** on the **Select Command** dialog box. After this, expand the User-Defined Commands node again.

Shell window

The Shell window provides access to the command line, where all utilities and processes are available.

You can also open multiple Shell Windows.

Site Daemons

Site Daemons are used to control access to engine files and to monitor engine processes and threads.

UNIX users

In UNIX, two site daemons are used for each site:

- Lock Manager controls access to files that are used by the engine. For example, error files, recovery files, SMAT files, and log files.
- Monitor Daemon monitors the state of engine processes and threads, and supplies that information to other tools. This watches the system and site conditions, and monitors pre-defined alert conditions.

Windows users

In Windows, only the Monitor Daemon is available.

Command-line users

- Lock Manager (UNIX only)
 - `lockmgr` accesses the Lock Manager.
 - `hcisitectl` turns the Lock Manager on and off.
 - `console -mp` queries the Lock Manager status.
 - `lm -mp` is the Lock Manager process.
- Monitor Daemon
 - Access the Monitor Daemon through `hcimonitor.d`.
 - `hcisitectl`, with appropriate flags, turns the Monitor Daemon on and off.

Site Daemons tool

Use the Site Daemons tool to start or stop any of the daemons.

- 1 Select the daemons to start or stop.
- 2 Select the alert configuration file.
- 3 Begin the action by clicking **Start Daemon** or **Kill Daemon**.

Caution: Always stop system processes before stopping the Lock Manager.

SMAT

The Saved Messages Administration Tool (SMAT) utility is for users to manipulate messages recorded by the engine, with these restrictions:

- Messages in a SMAT file can be deleted.
- Messages in the SMAT database cannot be deleted.
- Messages in both the SMAT database and SMAT file cannot be modified.

SMAT deals with all inbound and outbound messages that are saved through the Network Configurator. These are placed in two files collectively called the "saved message file."

The saved message file consists of these files:

- Index file (.idx)
This contains information about the messages.
- Message file (.msg)
This contains the body of the messages.

There is only one recovery database in a system site, but there are separate inbound and outbound saved message files for each connection.

The saved message file is written to the file specified in the network configuration file and is relative to `$HCISITEDIR/exec/processes/process_name`. Use the **Inbound** and **Outbound** tabs on the Network Configurator to specify where the engine saves the messages.

SMAT saves messages in a viewable message file. For interaction purposes, each message in the file is given a sequential number starting with zero. These sequence numbers are not used by other tools.

The SMAT files can be cycled based on the file size. Cycled files are kept as history data. You can configure the cycling options and history settings from the GUI.

SMAT files can also be cycled based on date and time. This is also available for cycling process logs.

See [SMAT history](#).

The current SMAT files are not compatible with earlier versions of the GUI. `hcismatconvert` is a tool that converts SMAT files between formats: old to new, or new to old.

See [hcismatconvert](#).

Symbolic-linked SMAT and SMAT database files outside of HCIROOT

You can move SMAT files to locations outside the Cloverleaf root directory. These files can be opened from any location.

See [Creating symbolic links](#).

Types of SMAT

You can select file-based SMAT or database-based SMAT.

File-based SMAT is configured through the SMAT GUI. This is maintained for backward compatibility.

Database-based SMAT is configured through the SMAT database GUI. The system engine stores SMAT messages in a SQLite database in an encrypted form. This tool implements cross-database search per the user's needs.

Message states

A message can be in one or more of three states:

- Not viewed
The message is not currently in the view list, although it is still in the saved message file and is available for viewing.
- Viewed
The message is currently in the view list. Select messages in the view list individually or as a group.
- Selected
The message is selected in preparation for an operation on the saved message file. After messages are selected, resend or remove them from the saved message file.

A message can be in several states at once. For example, a message that is being viewed can also be selected. A message can also be selected but not viewed. This would happen when a message has been previously selected, but the current display is another view list.

Saved message file usage

These are the uses of the saved message file:

- Outbound messages (OB), for disaster recovery on the remote host. For example, if the host loses delivered messages, the saved message file provides a method to resend the messages to the host.
- Inbound messages (IB), for message resends. These are inbound messages that must be resent through the engine as though they are being sent for the first time.

- Inbound and outbound messages (IB and OB), for historical records or statistical analysis. This provides the option of collecting information about specific connections, independent from any statistics kept by the system.

Both the engine and SMAT require access to the saved message files. After cycled, SMAT has access to previously saved data.

SMAT supports resending messages with metadata.

See [Resending messages](#).

See [Resending temp files](#) on page 228.

See [resend command](#).

See [Using resend_errordb and hcidbdump](#).

Understanding the current message

Information in the Current Message panel applies to the currently viewed message and the unfiltered contents of the message itself. Apply a filter to the body of the message to interpret it display.

When a new message displays, the statistics also change.

- **Current Message ID** shows the unique message identifier used by the system. A message ID consists of three parts: domain, hub, and site. In the system, the domain and hub are "0". The site is the sequential message number within this site.
- **Sending Date** shows the date and time this message was received by the engine from the originating host.
- **Sending Host** shows the name of the originating host.
- **Selected Encode Scheme** lists all of the encodings that are supported in the thread configuration, such as Latin-x, UTF-8, Big5, and so on. The selected encoding is used when displaying the currently selected message and when resending the modified message content.

Message data and metadata are stored in the error and recovery databases using UTF-8 encoding. The exception is error state 416: Inbound encoding conversion error and recovery state 1, on IB pre-TPS queue.

Error state 416 and recovery state 1 messages are the raw data received from the protocol driver.

Inbound SMAT files are a copy of the raw data from the protocol driver. This is primarily because inbound messages could be invalid. In this case, no Unicode representation can be obtained. Keeping inbound SMAT and recovery state 1 in raw form is important to make resending inbound messages operate correctly.

Outbound SMAT files are a copy of the post-TPS UTF-8 Unicode data. Outbound SMAT files also have a {ENCODING UTF-8} setting added to the .idx file for each message to distinguish them from earlier non-UTF-8 data.

- **Selected Encode Format** provides a choice of the type of formatting to use for the message file. Click the arrow to open a menu of choices.
- **Connection** shows the database connection list.
- For **Table Schema**, select a schema from the list of all currently imported schemas under the database connection.

- For **Detail level**, click the arrow to open a menu of detail levels for the output that is reported in the text area. Detail levels go from 0 (raw, unparsed data) to 4 (most detail).

Current message body

The body of the current message displays according to the format settings. If formatting is turned off, then the message body displays unformatted.

Fixed record formatting is accomplished with the record format utilities.

Because SMAT must run one of these utilities each time a message displays, it is much slower to move through messages with formatting turned on.

Turn formatting off to page through many messages when the record format is complex.

Scroll bar

The scroll bar shows the sequence number of the current message. It also shows how many messages are in the view list, and provides movement within the list.

The scroll bar always shows the sequence number of the current message within the saved message file, not the relative order within the view list. The scroll bar always goes from lower numbers on the left side to higher numbers on the right side. Note that gaps could be in the numerical sequence.

For example, if messages 21 and 26 are in the view list but not 22-25, the slider moves directly from 21 to 26 when moved one notch to the right.

A notable feature of the scroll bar is that SMAT does not redisplay the message body until the scroll bar is released. This is useful because formatting of fixed-record messages is a slow process. By not redisplaying messages until the scroll bar is released, this speeds up the process of moving to a specific message within the view list.

Editing a message

Click the **Edit** button to open the **Message Editor** dialog box, where you can edit a message before resending.

A copy is also made of the message; the original message is saved in SMAT.

Loading multiple files

Multiple files can be opened in one IDE instance.

Each tool instance can load an individual SMAT file. When a file is loaded in a tool instance, the tool instance shows the file name on the tool title. This specifies which file is loaded in this instance.

The SMAT files in the process directories within the current site are listed on the Site Manager into the SMAT category in individual process folders.

When a file is double-clicked, a SMAT instance is opened and the selected file is auto-loaded.

SMAT files are runtime data but not configuration objects. The initialization of the SMAT file list only happens when the Site Manager is initialized.

After initialization, the Site Manager does not reinitialize unless the Site Manager is refreshed.

View definition

You can apply search conditions directly to a view. The view definition is similar to a template that can be used to populate the search conditions when searching.

It is no longer necessary to separately configure view definitions. You can save the configured search conditions as a view definition as required.

Open the **Create a New View** dialog box by selecting **View > Create New View**.

Selecting **Add Specified Messages to View** and **Remove Specified Messages from View** open the same dialog box, the only exception being the name on the title bar.

You can use this dialog box to configure the search conditions from which to search the messages. You can save the configured search conditions as a view definition, or select a view definition from which to populate the search conditions.

Selecting a view

Use View opens the **Select View Definition** dialog box, where you can select a view definition from which to populate the search conditions.

This dialog box lists all view definitions in the SMAT file. When a view definition is selected, the Condition List table shows the search condition of the view definition.

Import and **Delete** are used to maintain the view definitions.

Importing view definitions

Import open the **Import View Definitions** dialog box.

Browse open a file browser that shows all available SMAT files that can be imported. The dialog box is used to select any SMAT file within the root.

After you select a SMAT file, the view definitions that are defined in the SMAT file are listed on the definition list of the **Import View Definitions** dialog box. Select a view and then click **OK** to import.

During importing, the view definitions may have name conflicts with existing files. When this happens, a message dialog box opens where you can decide whether to overwrite the existing definitions or cancel the import. The imported view definitions are shown on the view definition list.

Deleting view definitions

When a view definition is deleted, the next item in the filter list is auto-selected.

A confirmation dialog box opens before you can delete a selected definition.

Reloading a view

Click **Reload View** to reload the current view. The same view is kept if there are no changes to the file. If there are changes, then this updates the message count. Then the search criteria are applied to the new messages if the file has new messages since the last open.

This option reloads the same file, keeping the same view (search expressions, files, and so on). It also automatically moves the view to the last message.

This is useful if you are in a testing scenario and must repeatedly send a message through the interface to test code changes.

It is also useful to watch messages as they flow across the interface. For example, a message is sent every few minutes. You can click **Reload View** and see the new messages without having to reselect the file and reload the messages.

Condition list toolbar

The toolbar provides buttons for adding or deleting search conditions. Every search condition type has a corresponding button.

- **Add a Message ID Condition** specifies a range of message IDs.
 - **Add a Sending Date Condition** adds a Sending Date condition. You must specify **From** and **To** dates.
 - **Add a Sending Host Condition** specifies the host that sent the original message. Click the button to select from the menu.
 - **Add a Regular Expression Condition** is a way to express patterns that are sequences of these items:
 - Literal string
 - Matching character
 - Repetition clause
 - Alternation clause
 - Subpattern that is grouped with parentheses
- See [Using regular expressions for searching SMAT](#).
- The **Include** menu list gives you choices for specifying the search condition filter type added to the condition table:
 - Selecting **Include** includes all messages that match the search conditions.
 - Selecting **Exclude** excludes all messages that match the search conditions.

Condition list table

The Condition List table shows the currently configured search conditions.

For Table columns:

- **Condition Type** is a tree that represents the condition type and filter type of the search conditions.
- **Condition Value** shows the values of the search conditions.

For tree nodes, the Condition Type tree contains two levels of nodes:

- The first-level nodes in the tree are condition types such as Message ID, Sending Date, Sending Host, and Regular Expression.
- The second-level in the tree are the filter types such as Include and Exclude. The concrete condition values are attached to the Include or Exclude node.

When a condition is added, the table appends the condition to the specified condition type and filter type. If the nodes for the condition type and filter type are not there, then the table creates the nodes and adds the condition to them.

For sub-nodes, the Message ID and Sending Date condition values are ranges. Therefore, their condition value nodes have two sub nodes:

- **From** indicates the starting range.
- **To** indicates the ending range.

These nodes should be considered as a pair. You cannot delete only one of them. When the condition value node is viewable, these nodes are automatically expanded and viewable also.

Editing condition values

Edit a field by clicking under the **Condition Value**.

- Message ID and Regular Expression use an editable field.
- Sending Date uses an editable field and you can also open the Sending Date Editor.
- Sending host uses an editable field and a menu.

When creating a new condition, the default value is filled in the condition value.

- For the Message ID condition, the default value is 0.0.0 in both the From and To nodes.
- For sending date, the default value is MM/DD/YYYY 00:00:00.

To select a **host**, click under **Condition Value** to open a menu.

Date expression

The sending date value can be concrete timestamps or predefined date expressions. For example, `$Today`, `$Yesterday`, and `$DaysBefore`. To get the most recent data as the date moves ahead, use date expressions to reduce the instances that change the sending date values.

To configure the date value, click the button to open the **Sending Date Editor** dialog box.

This dialog box represents the current send date value. For the date value to be a date expression, select **Date Expression**. Then, open the menu to select an available date expression to use.

- `$Today`
- `$Yesterday`
- `$DaysBefore`

When the SMAT tool searches data, the date expressions are converted to the runtime date. The **Days** selector is enabled when you select the `$DaysBefore` expression. Then, you can specify the "before" days.

To use a concrete date, clear **Date Expression**. This enables the **Year**, **Month**, **Date**, **Hour**, **Minute**, and **Second** selection boxes from which to specify a concrete date.

SMAT tab on Client Preferences dialog box

These options in the **Client Preferences > SMAT** tab assist in view definition:

By default, these options are cleared.

When **Add All Messages to View automatically** is selected, all messages in the SMAT file are automatically added to the view after the SMAT file is opened. Otherwise, it opens with an empty view. Adding all messages to the view does not degrade the performance. It only changes the flags of the messages in memory and shows the first message in the view.

When **Restore Last View automatically** is selected, the last view definition that was recorded in the `ecd` file is automatically applied to the view. This might affect performance when loading a SMAT file. This happens because the entire message set in the SMAT file is searched using the defined conditions in the applied view definition.

Using regular expressions for searching SMAT

When using a regular expression to search for a particular message in SMAT, pay special attention to NL/CR.

The default search is case-insensitive.

The dot (.) matches all characters except line break characters, so ".*" only matches to the end of a line. At that time, you can use "?s". The dot implies the line separators.

For example:

```
MSH:~\&:HBOX:E:PCM:E:201208301315::ORU;R01;41:4299886:D:2.2:4299886: <CR>
PID:~01969187:01322089;;;E:"":TEST;BABY;BOY;"":":20110105:M:~7:250 FEDERAL PALZA;"": <CR>
OBR:1:00000002159E30015;HBOX:00000002159E30015;HBOX:30015;ABDOMEN SINGLE VIEW/KUB;RDE: <CR>
NTE:1:L:"" <CR>
OBX:1:ST:Exam32Room94;Exam Room:1:RM 3:::"":::: <CR>
```

When the content you search does not cross NL/CR, you can search the message that has both ORU and 2.2 in the body. To do this, use: `ORU.*2\2.`

When the content you search goes across NL/CR, you can search the message that has both ORU and OBR. To do this, use: `(?s)ORU.*OBR.`

To search the message that has both ORU and Exam, use: `(?s)ORU.*Exam.`

Note: Do not use `(.*\s)*` or `(.*\r)*`. There is a JDK bug where the matcher stays in an infinite loop and never returns the result.

Multi-line search

The SMAT regular expression search yields results if the pattern matches in one line.

For example, the search for `MSH.*PID` returns all HL7 messages containing both an `MSH` and `PID` segment. The search always returns no messages.

In this example, use `(?s)MSH.*PID` to use a regular expression to do a multi-line search .

The "s" flag makes the dot match any characters, including a line terminator.

Configuring message metadata

As part of the message, metadata is editable as the message content, with modified metadata being cached in memory for use when resending. The cached metadata is released when the current open SMAT file is closed.

On the SMAT GUI, click **Edit** to open the **SMAT Message Editor**, where you can edit the message content.

The left side of the dialog box contains read-only metadata and cannot be edited.

The right side contains editable metadata.

For the read-only and editable sides:

- The left-hand column is the metadata name.
- The right-hand column is the metadata value.
- All values are pre-populated with the metadata of the current message.

Editable metadata values are modified by clicking in the corresponding Value column, which makes the cell editable.

Modified messages are indicated by a red asterisk.

Cell editors

The cell shows an appropriate editor when it is editable.

- **Fixed set**
If the metadata value is a fixed set, the editor opens a menu of options.
- **Multiple Ones in a Fixed Set**
If the metadata value is multiple ones in a fixed set, the editor is a text field. There is also a button that opens a dialog box where you can select multiple items.
Clicking the **Value** button opens a dialog box for selecting multiple items from a set.
- **Flags cell editor**
This editor is similar to **Multiple Ones in a Fixed Set**. The text field is disabled because the Flags value is a hex number that indicates the kind of flags that are currently set up.
Clicking the **Value** button opens a dialog box for configuring the flags.
The read-only flags are not editable. Configure individual flags in the Editable Flags column.

In the Editable Flags column, `is_traced` is used to specify the tracing flag for resent messages. When the tracing flag of a resent message is set, all messages that result can be traced. This happens even when tracing for the threads through which they go is turned off.

Select the flags to include and click **OK** to apply the modifications. The selected flags are converted to a hex number and shown on the Message Editor.

- **Text editor**

For all others, the editor is a text field where you specify the correct values.

Metadata fields

This table lists all the metadata fields as they are stored in both message objects and in SMAT.

Not all metadata fields are stored in SMAT. Some fields are internal fields set by the engine that do not require to be stored in SMAT.

Message Object (Message.h)	SMAT (.idx)
<code>msg_type</code>	TYPE
<code>msg_class</code>	CLASS
<code>msg_flags</code>	FLAGS
<code>msg_state</code>	STATE
<code>msg_priority</code>	PRIORITY
<code>msg_mid</code>	MID
<code>msg_src_mid</code>	SRCMID
<code>msg_src_mid_group</code>	GROUPMID
<code>msg_src_thread</code>	SOURCECONN
<code>msg_orig_src_thread</code>	ORIGSOURCECONN
<code>msg_xlate_thread</code>	XLTHREAD
<code>msg_num_retries</code>	RETRIES
<code>msg_skip_xlate</code>	SKIPXLT
<code>msg_recovery_db</code>	USERRECOVERDB
<code>msg_group_id</code>	GROUPID
<code>msg_static_dirty_flag</code>	
<code>msg_routes</code>	
<code>msg_separator_chars</code>	SEPCHARS
<code>msg_dest_thread</code>	DESTCONN

Message Object (Message.h)	SMAT (.idx)
msg_orig_dest_thread	ORIGDESTCONN
msg_driver_control	DRIVERCTL
msg_record_format	DATAFMT
msg_user_data	USERDATA
msg_variable_dirty_flag	
msg_time_start_ib	TIMEIN
msg_time_start_xlate	TIMEXLT
msg_time_start_ob	TIMEOUT
msg_time_cur_que_start	TIMEQCUR
msg_time_total_que	TIMEQTOT
msg_time_recovery	TIMEREC
msg_time_stored	TIMESTOR
msg_time_archived	TIMEARC
msg_time_saved	TIME
context	SAVECONTEXT
dbTable	DBTABLE
siteName	
clVer	
hostname	
processName	

File synchronization

The SMAT tool checks if the currently open SMAT files are out-of-sync each time you attempt to access the files. If the files are cycled, then messages are deleted by other IDEs, or the file format has changed, SMAT opens an error message to confirm reopening the file.

Select **Yes** to reload the files.

Resending temp files

When the user edits message contents before doing a resend, the new message is saved to a `temp` file. This is used in the `resend` command.

If multiple messages are being resent, then the metadata `temp` file consists of many lists.

If necessary, then you can edit metadata and message content before doing a resend.

The edited metadata is stored in a separate temporary file. This file's format is based on the index file format, with two differences:

- It has no requirement to be of fixed length. `SEPCHARS`, `DRIVERCTL`, and `USERDATA` contains the actual variable-length contents of those fields from the `.msg` file instead of length values.
- Many fields that are stored in SMAT are not required in the `temp` file and are omitted. They include the fields for context, type, and priority. These are specified as arguments to the `resend` command and many fields that are rewritten by the engine.

See [Using resend](#) and [Using resend_errordb and hcidbdump](#).

resend command

SMAT handles resending using the `resend` command through `hcicmd`.

```
hcicmd -p process -c "thread_name resend <ib
[dest_thread]|ob> data|reply priority msgfilename
len10|nl|eof [metafilename]"
```

The optional `msgfilename` argument to the `resend` command is used for metadata resending.

- `msgfilename` is an optional argument. If this argument is not supplied, then the command works the same as version 5.7.
- `msgfilename` is the temp file that stores the message contents, which is unchanged. A new temp file is used to store edited metadata. This is referenced by `msgfilename`. The new temp file is deleted after the resend is complete.

Resending involves two temp files. Correct functioning depends on the number of messages in the message temp file matching the number of metadata entries in the metadata temp file. To assure that corrupt data is not sent in the case of a mismatch, `resend` counts the number of messages and corresponding metadata in both files. This is performed before running the `resend` command. No messages are resent in the case of a mismatch.

Specifying the destination threads when resending messages

The uniform destinations for all resent messages can be specified on the Network Monitor and SMAT Resend dialog boxes. For the reply type, multiple destinations are not supported.

For other editable metadata, they can be the same as the destinations. Destinations in the metadata file are overridden by `dest` from the command line.

Note: When specifying the destination thread for the `resend`, inter-site threads are not supported.

On the command line, the resend command takes *dest_for_ib* as the option between Location and Type:

```
resend ib_pre_tps|ib_post_tps|ob_pre_tps|ob_post_tps [dest_for_ib] data|reply priority filename
len10|nl|eof [metafilename]
```

Examples

```
hcicmd conn_6 resend ib_pre_tps {conn_1 conn_3} data 5120 d:\\-A.txt nl
```

This places two messages from file `d:\\-A.txt` into the `ib_pre_tps` data queue.

```
hcicmd conn_6 resend ib_pre_tps {conn_1 conn_3} datac 5120 d:\\-A.txt nl
```

Response:

Type must be data or reply

```
resend ib_pre_tps|ib_post_tps|ob_pre_tps|ob_post_tps [dest_for_ib] data|reply pri filename
len10|nl|eof [metafilename]
```

```
hcicmd conn_6 resend ib_pre_tps {conn_1 conn_3} datac 5120 d:\\-A.txt nlx
```

Response:

Incorrect options: Stylelen10|nl|eof OR Typedata|reply

```
resend ib_pre_tps|ib_post_tps|ob_pre_tps|ob_post_tps [dest_for_ib] data|reply pri filename
len10|nl|eof [metafilename]
```

resend_errordb and hcidbdump commands

The ability to resend with modified metadata also applies to error database resending. Error database resending is performed through `hcicmd` with the `resend_errordb` command:

- `resend_errordb mid`
- `resend_errordb mid file name len10|nl|eof`

If only the *mid* is supplied, then the message with that *mid* is resent directly from the error database with preserved metadata.

If three arguments are provided, then the message from the file is sent with metadata from the error db for the given *mid*. This *mid* argument supports a range of mids in the same format as `hcidbdump`, and supports resending multiple messages using this mid format:

```
x.y.z = only this message
```

```
x.y.z1:x.y.z2 = x.y messages z1 through z2
```

These options takes two arguments:

```
resend_errordb -m metafilename
```

This resends from the error database, but with modified metadata from a file. This file takes the same format as the SMAT metadata temp file.

The mids that are used in the resend come from the metadata file, meaning that when using metadata, `resend_errordb` also supports resending multiple messages.

Another option takes four arguments:

```
resend_errordb -m metafilename msgfilename len10|nl|eof
```

This performs an error database resend using both message data and metadata from files. As with `resend`, if the number of messages in both files do not match, the command returns an error without resending any messages.

If two or four arguments are used, but the first argument is not `-m`, then the command returns an error.

As `resend_errordb` takes from one to four arguments, the full form that is called from `hcicmd` is as follows:

```
hcicmd -p process_name -c ". resend_errordb mid"
hcicmd -p process_name -c ". resend_errordb -m metafilename"
hcicmd -p process_name -c ". resend_errordb mid msgfilename len10|nl|eof"
hcicmd -p process_name -c ". resend_errordb -m metafilename msgfilename len10|nl|eof"
```

-n metadata_filename option

To simplify these options, `hcidbdump` enables dumping metadata from the error database into a file of the necessary format. The option for this is `-n metadata_filename`.

```
hcidbdump -n metadata_filename -e -m mid output_filename
```

For example:

```
hcidbdump -n meta.txt -e -m 0.0.1:0.0.9999 out.txt
```

This generates the metadata file (`meta.txt` in the above example) which you can modify, use as the second argument to `resend_errordb`.

SMAT encoding and temporary files

In encoding persistence, the **Selected Encode Scheme** and **Selected Encode Format** settings are designed to be applied to a SMAT working session. After they are set, they affect all the messages displayed in SMAT. These settings persist for each SMAT file, to eliminate the requirement to reset the values each time you view the same set of messages.

The type of character encoding and formatting for displayed messages are selected from menus.

The encoding scheme and format apply to all displayed messages. This information is stored in a separate file with an `.ecd` extension for each viewed SMAT file. Each SMAT file can then have its own encoding settings which are applied automatically when messages of a SMAT file are reloaded and viewed.

The `.ecd` file is a text file that contains the encoding scheme and format for the SMAT `.msg` and `.idx` files with the same name.

The contents of the `.ecd` file look similar to:

```
[encode_scheme]
encode_scheme=<encode_scheme_value>
[encode_format]
encode_format=<encode_format_value>
[format_options_frl]
frl_file=<frl_file_name>
frl_detail_level=<frl_detail_level_value>
[format_options_hl7]
hl7_version=<hl7_version>
hl7_variant=<hl7_variant_name>
hl7_show_field_names=<hl7_show_field_names_value>
hl7_detail_level =<hl7_detail_level>
[format_options_hprim]
hprim_version=<hprim_version>
hprim_variant=<hprim_variant_name>
hprim_show_field_names=<hprim_show_field_names_value>
hprim_detail_level =<hprim_detail_level>
[format_options_hrl]
hrl_file=<hrl_file_name>
hrl_detail_level=<hrl_detail_level_value>
. . . .
```

In the file contents:

- Brackets (`[]`) contains section comments.
- All values in `< >` are replaced by real values in an `.ecd` file.
- Every field has its own default value.

The default value is assumed in these situations:

- The `.ecd` file is missing.
- The `.ecd` file exists, but the field setting is not found.
- The `.ecd` file exists, and field setting is found with empty value.
- The `.ecd` file exists, and field setting is found with non-recognized value.

Message encoding

The default message encoding is recorded into the SMAT file so that the SMAT tool can default to it when the SMAT file is opened:

- For inbound messages, the default message encoding is the thread encoding.
- For outbound messages, the default message encoding is UTF-8.

In earlier versions, when a SMAT file was opened for the first time, the encoding defaulted to ASCII until changed by the user. After this was set, subsequent openings of the same file used the user's encoding that is saved in the `ecd` file.

Now, opening a SMAT file checks if an `ecd` file exists. If it exists and the default encoding is found but the user encoding is not found, then the SMAT tool uses the engine default encoding. This encoding is saved as a user encoding in the `ecd` file. If both the engine default encoding and user encoding are found in the `ecd` file, then the user encoding is used.

This process shows the encoding defaults:

- 1 Inbound message arrive with some encoding, for example, latin1.
- 2 This is saved to the inbound SMAT in the same encoding in which it came, for example, latin1.
- 3 The engine uses UTF-8 internally, so that the message is converted to UTF-8 in the engine.
- 4 The message is written to the outbound SMAT in UTF-8, so that a resend sends in UTF-8 to the engine.
- 5 The actual message going from the engine through the outbound protocol is what is specified in the outbound protocol thread, for example, latin1.

Encode scheme values

`encode_scheme_value` stores the user-specified value for the encode scheme. SMAT reads the value to decide which encode scheme to use when displaying a message.

There are three possible values and corresponding IDE choices:

- (Default value) ASCII no encoding is performed (`enc_ascii`)
- EBCDIC use defined mapping (`enc_ebcdic_hie`)
- EBCDIC use alternate mapping (`enc_ebcdic_alt`)

Encode format values

`encode_format_value` stores the user-specified value for the encode format. SMAT reads the value to decide which encode format to use when displaying a message.

For format options, the IDE reconfigures according to the **Selected Encode Format** (`encode_format_value`): FRL, HL7, and so on.

Note: When you first start, all format values are default values.

Client memory cache

SMAT builds a cache mechanism to reduce the frequency of reading and writing the `.ecd` file.

In these circumstances, SMAT directly reads from and writes to the `.ecd` file for the persisted information:

- The SMAT utility is closed
- The SMAT file is loaded

When you change the settings of **Selected Encode Scheme** or **Selected Encode Format**, option information is stored in the client memory cache.

Index file

When a SMAT index file is loaded, the IDE attempts to locate the corresponding `.ecd` file in the same directory as the index file. If the `.ecd` file does not exist, then the IDE creates a new `.ecd` file for the loaded index file with the same name.

After IDE loads the `.ecd` file successfully, the `.ecd` file is locked. The default options are also passed to the client to create the initial cache and are also used to display messages.

When the SMAT index file is closed, the IDE reads the persisted information from cache and store them into the `.ecd` file. The `.ecd` file is now unlocked.

Temporary files

There is no requirement to manually remove temporary files after resending.

After resending messages, temporary files that are generated during resending messages are automatically removed. The **Result** dialog box informs you that the temporary files are deleted.

Saved message format

The saved message file consists of index and message files.

Index file

The index file is a Tcl keyed list. The keyed list consists of a set of key-value pairs, where:

- Each keyword is followed by a value.
- The entire key-value pair is enclosed in { } (braces).
- Each line in the index file describes exactly one message in the message file.

The exact interpretation of a message depends on the record format, which the user must know. SMAT has no way of obtaining that information.

With the exception of `DATAFMT`, which is not used by the engine but can be set by the user, these fields have maximum lengths. They are made fixed-length by padding with white space. Fixed-length fields are used by the GUI to search the index by offsets instead of having to load the entire index into memory.

Message file

SMAT does not interpret the body of the message, though it does let the user format the message body as an aid in manual searches.

In addition to messages, the message file also contains variable-length metadata, which is a raw string to be parsed using offsets from the index file.

This table shows the offsets that are used to parse the `.msg` file:

Field	Description
OFFSET	Pointer to the message beginning
LENGTH	Message length in bytes
DATAFMT	Data format metadata length
SEPCHARS	User-defined separators metadata length
DRIVERCTL	Driver control metadata length
USERDATA	User data metadata length

Fixed length fields

This table shows the fixed-length fields. These fields are added as the length in bytes of the fixed-length metadata in the .msg file.

Field	Description
MID	Message ID
SRCMID	Source message ID
TYPE	Message type; DATA OR REPLY
SOURCECONN	Source connection
ORIGSOURCECONN	Original source connection
TIME	Time stamp of message entering SMAT
PRIORITY	Message processing priority; valid range is 4096-8192 (5120 is the default)
DATAFMT	Data format
SAVECONTEXT	Message that is saved on inbound read or outbound write
OFFSET	Pointer to the first character of message in .msg file
LENGTH	Length of message in .msg file, in bytes
FLAGS	Flags
STATE	Recovery/error state
GROUPMID	Parent MID for group
XLTHREAD	Thread that translated the message
RETRIES	Number of proto write attempts
SKIPXLT	Does this message get translated?

Field	Description
USERRECOVERDB	State of message in recovery db
GROUPID	For group control in TPS
TIMEIN	Inbound latency start
TIMEXLT	Xlate latency start
TIMEOUT	Outbound latency start
TIMEQCUR	Start time on current queue
TIMEQTOT	Total time on all queues
TIMEREC	Last recovery db update
TIMESTOR	Store message to message archive db
TIMEARC	Archive the message to external db
DBTABLE	Database table for external database

Variable length fields

This table shows the variable-length fields. These fields are added as the length in bytes of the variable-length metadata in the .msg file.

Field	Description
DESTCONN	Destination connection
ORIGDESTCONN	Original destination connection
DATAFMT	Formerly always null
SEPCHARS	User-defined separators
DRIVERCTL	Keyed list for PD
USERDATA	Data added by user

In the .idx file, between each full entry is a newline character (`\n`). There is no carriage return (`\r`).

In an actual index file each entry is a single line, so an index file with three entries would look similar to:

```
{{MID  {{DOMAIN 0  }} {HUB 0  }} {NUM 1407  }} ...
{{MID  {{DOMAIN 0  }} {HUB 0  }} {NUM 1410  }} ...
{{MID  {{DOMAIN 0  }} {HUB 0  }} {NUM 1413  }} ...
```

SMAT history

When SMAT files exceed a specified maximum size, SMAT cycling is used to back up the files.

You can also cycle SMAT using date and time. This feature is also available for cycling process logs.

The backed up files have the same file names but have a time stamp that makes sure the files are unique and identifiable. The SMAT file history utility is used to maintain the cycled SMAT files.

Both SMAT file and SMAT database files are managed by SMAT history.

Cycling is accomplished running this command-line:

```
<conn> save_cycle {in|out}
```

Process Configuration dialog box

On the **Process Configuration** dialog box, select **Automatic SMAT Cycling** to specify the SMAT files are cyclable. By default, the option is not selected and the SMAT files are not cycled.

The **Threshold in kilobytes** field specifies the maximum file size before cycling. This applies only to .msg files. When the threshold value is zero, the cycling mechanism is disabled, even if the option is selected.

SMAT files can also be cycled based on date and time. This is also available for cycling process logs.

Automatic SMAT cycling and purging

Automatic SMAT cycling happens when the cycle threshold is reached.

Inbound SMAT is cycled if the inbound protocol encoding configuration is changed and takes effect.

The SMAT Database cycle threshold is based on the size of messages, not the size of .smatdb.

As SMAT is locked when performing a search, this could cause cycling to fail when SMAT cycling is triggered. In this case, messages are written to the current SMAT so that they are not lost. After the IDE releases the lock, the cycle is successful when it is triggered again.

How and when purging happens

In this example, a script has been created that explains how the auto SMAT cycling and purging works. The script estimates maximum sizes for a site based on their settings and number of threads that SMAT has enabled.

Purging points:

- **Delete the oldest SMAT history file when history file's total size for a process exceeds number KB** applies to the entire `SmatHistory` directory. This is the overall size of the directory, not per SMAT or thread. This value is the threshold of the total size of SMAT files within the `SmatHistory` folder. When the cycled file is backed up to the `SmatHistory` folder, the engine calculates the sum size of all the files. The engine

then compares to this threshold. After the sum exceeds the threshold, the oldest file is deleted. The timestamp is also the judgment.

This option is process-level effective. The total size contains all differently-threaded SMAT files; it does not indicate the total size of special thread SMAT files.

- **Delete the oldest SMAT inbound/outbound history file when history file's number for a thread exceeds number files** is for each direction (in/out) per thread.

This value controls the maximum number of SMAT files within the `SmatHistory` folder. When the cycled file number exceeds this threshold, the newest cycled file is saved and the oldest file is deleted. The time stamp is the judgment to decide which file is the oldest and which is the newest.

This option is thread-level effective. Each thread's SMAT file number limitation is the threshold.

- **Delete SMAT history file for a process when it is older than number days** purges after the file is *number* of days old.

If multiple options are provided, then the options are "OR". This indicates whichever condition is true is used.

Age means the time stamp of the cycled SMAT file.

This option keeps saving the cycled files as long as the value is set. When a new cycled file moves into the `SmatHistory` folder, the engine individually checks all files by time stamp. When a file's saving time exceeds this threshold, the file is deleted.

If all files are set a value, then the engine deals with them using the priority **maximum number of SMAT history > maximum size of SMAT history > maximum age of SMAT history**.

- **Maximum Logs Total Size** is for the entire `LogHistory` directory. This is the overall size of the directory and does not take into account the number of files.
- **Maximum Number of Log Files** counts each file, so `.err` and `.log` files are counted. In this case, you double the number of log files you require.

If multiple options are provided, then the options are "OR". this means whichever condition is true is used.

MonitorD cycling uses the **Purge** settings for `LogHistory`.

Example:

```
SMAT History Max Files for Site helloworld - 5
SMAT History Max Size for Site helloworld - 500.000 MB
Log History Max Files for Site helloworld - 50
Log History Max Size for Site helloworld - 100.000 MB
Process crnl_ibparse_append has 4 SMATs cycling at 100.000 MB
Process crnl_ibparse_append Estimated SMAT Max Size use 500.000 MB - Max number of SmatHistory files: 5
Process crnl_ibparse_append Estimated SMAT Max Number of Files use 2.000 GB
Process crnl_ibparse_append Estimated Log History Max Size use 100.000 MB
Process crnl_ibparse_append Estimated Log History Max Files use 250.000 MB
Process crnl_ibparse_rewrite has 4 SMATs cycling at 100.000 MB
Process crnl_ibparse_rewrite Estimated SMAT Max Size use 500.000 MB - Max number of SmatHistory files: 5
Process crnl_ibparse_rewrite Estimated SMAT Max Number of Files use 2.000 GB
Process crnl_ibparse_rewrite Estimated Log History Max Size use 100.000 MB
Process crnl_ibparse_rewrite Estimated Log History Max Files use 250.000 MB
Process crnl_ibtps has 4 SMATs cycling at 100.000 MB
Process crnl_ibtps Estimated SMAT Max Size use 500.000 MB - Max number of SmatHistory files: 5
Process crnl_ibtps Estimated SMAT Max Number of Files use 2.000 GB
Process crnl_ibtps Estimated Log History Max Size use 100.000 MB
Process crnl_ibtps Estimated Log History Max Files use 250.000 MB
Process crnl_oob has 4 SMATs cycling at 100.000 MB
Process crnl_oob Estimated SMAT Max Size use 500.000 MB - Max number of SmatHistory files: 5
```

```

Process crnl_oob Estimated SMAT Max Number of Files use 2.000 GB
Process crnl_oob Estimated Log History Max Size use 100.000 MB
Process crnl_oob Estimated Log History Max Files use 250.000 MB
Process eater_in_pdl has SMAT Cycling turned off or no threads using SMAT
Process eater_in_pdl has Log Cycling turned off
Process eater_in_tcp has SMAT Cycling turned off or no threads using SMAT
Process eater_in_tcp Estimated Log History Max Size use 100.000 MB
Process eater_in_tcp Estimated Log History Max Files use 250.000 MB
Process feeder_in_pdl has SMAT Cycling turned off or no threads using SMAT
Process feeder_in_pdl has Log Cycling turned off
Process feeder_in_tcp has SMAT Cycling turned off or no threads using SMAT
Process feeder_in_tcp Estimated Log History Max Size use 100.000 MB
Process feeder_in_tcp Estimated Log History Max Files use 250.000 MB
Process feeder_out_pdl has SMAT Cycling turned off or no threads using SMAT
Process feeder_out_pdl has Log Cycling turned off
Process feeder_out_tcp has SMAT Cycling turned off or no threads using SMAT
Process feeder_out_tcp Estimated Log History Max Size use 100.000 MB
Process feeder_out_tcp Estimated Log History Max Files use 250.000 MB
Total unique SMATs (in/out) for site helloworld - 16
Total Max Estimated SMAT Size use for site helloworld - 2.000 GB
Total Max Estimated SMAT Files use for site helloworld - 8.000 GB
Total Max Estimated Log Size use for site helloworld - 700.000 MB
Total Max Estimated Log Files use for site helloworld - 1.750 GB

```

Site Preferences dialog box

The SMAT file history settings are stored in the `siteInfo` file. This setting is configured on the **Site Preferences > SMAT** tab. On this tab, you can enable the SMAT History feature.

The **Delete the oldest SMAT inbound/outbound history file when history file's number for a thread exceeds number files** and **Delete the oldest SMAT history file when history file's total size for a process exceeds number KB** options are used to maintain the SMAT history files. When **Unlimited** is selected, the cycled SMAT files are saved. Otherwise, the earlier files are deleted after the limitations of the file number or the disk size are reached.

Compress Command is the command to compress the backed-up SMAT files to save disk space.

- If there are more than one file to compress, then each file is compressed individually.
- If no command is configured, then the cycling files are not archived.

By default it is set to NULL and compression is disabled. The `gzip` and `compress` commands are supported.

SMAT cycles message, index, and `.ecd` files with these naming conventions:

- `filename.old.msg`
- `filename.old.idx`
- `filename.old.ecd`

To avoid "clobbering", unintentionally overwriting, earlier cycled files, the `SmatHistory` feature cycles the files with a time stamp. This feature is similar to the engine's `LogHistory` feature.

When this feature is enabled, it cycles the old files to the `process directory/SmatHistory` directory instead of straight to the process directory.

The cycled files have these file names:

- `filename.YYYY-MM-DD_HH-mm-SS.msg`
- `filename.YYYY-MM-DD_HH-mm-SS.idx`
- `filename.YYYY-MM-DD_HH-mm-SS.ecd`

In these files:

- `YYYY` is the year
- `MM` is the month
- `HH` is the hour
- `mm` is the minute
- `ss` is the second

For example, files named `conn_1.msg` and `conn_1.idx` are cycled 42 seconds after 2:03 PM on January 21, 2009. The message and index file names are:

```
conn_1.20090121140342.msg  
conn_1.20090121140342.idx
```

They would then be placed in the `SmatHistory` directory.

Differences with earlier versions

Previous versions would have named the above files as:

- `conn_1.old.msg`
- `conn_1.old.idx`

When the **SmatHistory** feature is enabled, the files are cycled with the `SmatHistory` convention, begun in version 6.0 and later.

When the **SmatHistory** feature is disabled, the files are cycled with the `.old` convention of earlier versions.

See [hcismatconvert](#).

The **SmatHistory** feature settings are contained in the `siteInfo` file.

SMAT secondary

SMAT secondary provides HA capability to the SMAT subsystem. In the event that the SMAT subsystem cannot write to the primary file or database, it creates a secondary file or database. Messages are then written to the secondary file/database.

The SMAT subsystem continues in this manner until the primary is available again. At that point, the secondary data is collapsed into the primary and new messages then continue to be written to the primary. Collapsing from secondary to primary happens every two minutes.

The SMAT subsystem cannot write to the primary under these situations:

- The IDE has SMAT open when the engine is writing to it.
- The IDE is reading from SMAT when the engine is writing to it.

Saved message file management

To manage saved message files, SMAT provides an interface for access and modification. These are options for resending and removing messages:

- Resend messages to a currently running thread inside the engine, or to a length-encoded file. After saved to a file, resend the messages to a thread at a later date or modify with a text editor.
To resend messages from multiple connections, work on each saved message file sequentially, because each connection saves its messages to another file.
- If the saved message file is used only for disaster recovery, then select a period of time to keep delivered messages when removing earlier messages.
- If the saved message file is used for statistical purposes, then it is useful to delete messages that are processed by the user-written statistics program.

Configuring access to saved message files

Both the engine and SMAT require access to the saved message files. To configure this:

- 1 Open the Network Monitor.
- 2 Click the view in the View List pane.
- 3 Right-click the thread icon on the **View** tab to open the context menu for thread management.
- 4 Select **Control > Full** to open the **Thread Controls** dialog box.
- 5 Then, click **cycle save**. This opens the **Data Direction** dialog box.
- 6 Select whether inbound or outbound saved messages are to be cycled:
 - Select **In** for all inbound saved messages contained in `.idx` files to be cycled.
 - Select **Out** for all outbound saved messages contained in `.idx` files to be cycled.
- 7 Click **OK**. This causes the current saved message file to close, be renamed with an `.old` extension, and then opens a new saved message file. This gives SMAT access to the previously saved data.

Cleaning out the saved message file

This is the most common method for cleaning out the saved message file.

- 1 Open SMAT and load the view to clean out.
- 2 Select **View > Remove Specified Messages from View** to create a view list that represents the messages to remove.
- 3 Configure the view with new values, and click **Apply**.

Resending messages

For information on the web service APIs, see:

- [SMAT message resend related web services](#)
 - [Error message resend related web services](#)
- 1 Select **Selected > Resend** to open the **Resend Message** dialog box.
Use this dialog box to resend messages in the selection list to a length-encoded file or to a running engine thread. Messages can be sent as inbound or outbound, and as data or reply type. Click **Apply** to resend.
Unselected > Resend resends messages that are not in the selection list.
 - 2 Select **Thread** or **File** from the **Resend n Selected Message To** menu, where n is the number of selected messages.

Destination threads

The **Destination Threads** field is used to specify the destination threads for message resends.

Note: When specifying the destination thread for the resend, inter-site threads are not supported.

Destination threads are relevant to the source thread and message type:

- **Reply Type:** Only those threads defined as reply route destinations of the source thread can be specified as the destination threads.
- **Data Type:** Only those threads defined as route destinations of the source thread can be specified as the destination threads.

List is unavailable in these conditions:

- **Reply Type** is selected and the selected source thread does not have the reply route destinations.
- **Data Type** is selected and the selected source thread does not have the route destinations.

Click **List** to open the **Select Destination Threads** dialog box, where you can select the destination threads from a thread list.

- If **Reply Type** is selected, then the dialog box lists the available reply route destination threads for the source thread.
- If **Data Type** is selected, then the dialog box lists the available route destinations threads for the source thread.

Note: If the resend Context is Outbound pre-TPS or Outbound post-TPS, then Destination Threads and List are unavailable.

SMAT message encoding options

SMAT records the message encoding as the default encoding in the `ecd` file.

The default message encoding is recorded into the SMAT file so that the SMAT tool can default to it when the SMAT file is opened.

- For inbound messages, the default message encoding is the thread encoding.
- For outbound messages, the default message encoding is UTF-8.

The SMAT tool shows the default encoding when the SMAT files are opened for the first time.

The **Encoding** list is used to specify the encoding for resent messages. The user-selected encodings are passed from the list to the `resend` command line.

The default value of the list in the **SMAT Resend Message** dialog box is derived from these steps:

- The default encoding is used, if there is one; if not, then . . .
- The NetConfig thread encoding is used, if there is one; if not, then . . .
- The user-selected encoding from the menu is used.

When a SMAT file is opened, it checks if an `ecd` file exists.

- If an `ecd` file does exist and the default encoding is found but user encoding is not found, then SMAT uses the engine default encoding. The encoding is saved as user encoding in the `ecd` file.
- If both engine default encoding and user encoding are found in the `ecd` file, then the user encoding is used.

See [SMAT encoding and temporary files](#).

Resending messages with metadata

- 1 Open the **Resend Message** dialog box to resend messages to threads with specified metadata.
- 2 To resend messages with their own metadata, select **Include Metadata**.
The message content, message type, and priority are applied to all messages when resending. This happens even when the check box is selected, to keep all resent messages uniform.
- 3 Click **Apply** after configuring the resend. The **Result** dialog box opens with the resend results.

Message formats

The SMAT utility provides maintenance utilities to process all saved inbound and outbound messages.

On the Inbound pane of the Network Configurator's **Properties** tab, there are two options: **Save inbound messages** and **Save outbound messages**.

When one of these is selected, messages can be viewed, modified, deleted, and resent using the SMAT utility.

You can view these messages in SMAT using encoded formats such as CR>NL and FRL. You can also use other formats such as EDIFACT, VRL, HRL, HL7, LDL, HPRIM, NCPDP F&B, NCPDP SCRIPT, NCPDP Telecom, VRL, X12, and XML. The default value is None.

The SMAT pane changes according to the selected format.

- **CR>NL**
When **CR>NL** or **None** is selected, the Format Options pane is hidden. The other formats have other options, according to the selected format.
- **Detail Level**
This is the detail level of the message formatting. Detail levels go from 0 (raw, unparsed data) to 4 (most detail). This option is not available when **CR>NL** or **None** is selected.

Monitoring parsing progress

All of the indexes in the index file are parsed after being loaded. A progress bar in the status bar shows the parsing progress.

The progress bar is created when an index file starts to load, continues throughout the loading progress, and hides when it is finished.

It is hidden when the active tool is not SMAT. It is shown again when the active tool is changed back to SMAT if the loading does not finish.

The index loading is interrupted if you load another SMAT file before the loading is completed.

The progress bar is also shown when messages are deleted, resent, or searched, and continues showing progress until completion. The progress bar is hidden when the process is over.

All SMAT operations are blocked after the parsing progress is started and before it is completed. You can still access other IDE features without waiting for the loading to complete.

The parsing status is checked before a view is created (for example, when you select **Add All Messages To View**). If parsing is not finished, then a message dialog box opens informing you that the index is still parsing and to try again.

SMAT database

In the SMAT database GUI, the system engine stores SMAT messages in a SQLite database in an encrypted form.

SMAT database timestamps are in milliseconds. This adds a finer granularity when, for example, you write a widget to calculate latency or processing time.

Time-related fields have 13 digits. Because the feature bitmap is set, there cannot be two different types of time format in one SMAT database.

The top 58 bits are the feature bitmap.

The lowest bit of the 58 bits is “1”, which means milliseconds is implanted.

The lower 8 bits are the SMAT database version.

The `smat_info.version` field contains 0x102(=00000001 00000010b).

The database is cycled if the earlier version is detected.

Starting a search

Note: During a SMAT Database search, the SMAT Database tool is blocked. This indicates that no other function can be performed during a search.

When starting a search, you select the content to search for.

This table shows the **Content** menu options:

Search option	Description
HL7	By default, the auto-complete database uses the segment list from the HL7 2.7 standard. To use a variant (for example, to search on a Z segment field and have auto-complete), run <code>hcicreateformatdb</code> . This should be run after an HL7 variant Save operation.
HPRIM and HL7+HPRM	These are intended to specify the formats of the messages that the content search is applied on.
regular expression	<p>When you change from the other options to regular expression, the field populates with the regular expressions that are pre-built from the Field/Alias search expression. You can then modify the regular expression.</p> <p>Refresh Search is enabled when this option is selected for the search content.</p> <p>This performs an instant content search for all message formats, and a way to view what was created by the HL7/HPRIM regex builders.</p> <p>If you create an ad-hoc regex and switch the context back to the HL7/HPRIM options, then it is unavailable.</p> <p>If the regular expression is modified, then when you select HL7 a confirm dialog prompts whether to save the modified regular expression as a search criteria. Selecting Yes inserts the modified regular expression into the Advanced Criteria table as a regular expression criteria.</p> <p>Regular expression syntax is based on PCRE (Perl Compatible Regular Expressions).</p>

Entering text into the **Content** field opens a list of auto-complete choices.

You can search messages with a field/alias. The search supports auto-complete and instant searches.

When changing from a regular expression back to the format specific one:

- If the pre-built regular expression is not edited in the text field, then the text field shows the original field/alias search expression.
- If the regular expression has been edited, then a warning message is displayed, This gives you an opportunity to save it to advanced criteria before clearing the text field.

`hcicreateformatdb` is automatically called when saving a variant through the HL7 GUI or deploying a BOX with HL7 variants included.

If the variant is changed outside the GUI, then you must use `hcicreateformatdb` to add the variant to the **Auto-Complete** menu.

After configuring your search parameters, click **Save Criteria**. This opens the **Save Criterion Definition** dialog box. The current criteria are saved in `$HCISITE/smatdb/criteria_defs/criteria_def_username.ini`.

Canceling a search

In some instances, you could accidentally start a SMAT Database search for a large amount of files. When this happens, you can cancel the search after it has started.

The **Search** and **Cancel** functions share the same button.

Before a search is initiated, the button is labeled **Search**. After clicking **Search**, the button label changes to **Cancel**.

After the message search has completed, the button label changes back to **Search**.

Symlinking and NFS file systems

SMAT Database is designed for local file systems.

Symlinking should not be used, especially with NFS (Network File System) file systems. In this scenario, SMAT Database is implemented with SQLite database, but SQLite database has known locking control difficulties on NFS file systems. The SQLite database is prone to corruption on NFS file systems. You should avoid using SMAT Database on NFS file systems or symlinking SMAT Database.

Advanced criteria

Additional search criteria are configured in the Advanced Criteria pane.

By default, this pane is hidden. Click the **Expand/Collapse Advanced Criteria** button to display it.

Clicking **Add Criterion** opens the **Add a Search Criterion** dialog box.

When **Message Metadata** is selected, most metadata can be configured to be metadata criterion.

When **Message Content** is selected, **Operator** is **regular expression**. Specify the search value in **Value**.

SMAT database cycling

The `SiteInfo` file is located at `integrator\siteProto\SiteInfo`. In this folder are `smathistorylinkpath` and `smathistorybasethreadname`.

`smathistorylinkpath`

`smathistorylinkpath` represents the link path. Ensure you have the ACL (Access Control List) to the link path.

Example:

- On Linux, `smathistorylinkpath=` is `smathistorylinkpath=/The/real/path`.
- On Windows, `smathistorylinkpath=` is `mathistorylinkpath=C:\The\real\path`.

`smathistorylinkpath` is the string value that identifies the path where `SmathHistory` must be linked. Ensure `hciengine` has access permission. If it does not, then the link fails.

smathistorybasethreadname

`smathistorybasethreadname` represents whether `SmathHistory` is subdivided based on thread names.

`smathistorybasethreadname` identifies whether `SmathHistory` is subdivided based on thread names. If enabled, then this indicates it is applied to the scope of the site. All processes are included.

- "0" indicates that this is the default value.
- "1" indicates that `SmathHistory` is subdivided base on thread names.

Database cycling links

See [Symbolic links](#) on page 67 for additional information on configuring the links for database cycling.

SMAT Database auto-cycling based on age

These options are available for SMAT DB auto-cycling:

- At the process level, an option to automatically cycle `smat` files based on a "time interval":
 - Key: `SMATCYCLETIME`
 - Unit: Hour
 - Default value: 24 hours (1 day)
- At the site level, an option to move history files older than "X" days to a specific folder. Default value: 7 days.
 - Key: `SMATHISTORYARCHIVEAGE`
 - Default: 0
 - This disables archiving the older history file.
 - Unit: Day
 - Default: 7 days
 - Range: > 1
 - If `SMATHISTORYMAXAGE` is set, the `SMATHISTORYARCHIVEAGE` must be less than `SMATHISTORYMAXAGE`.
 - Key: `SMATHISTORYARCHIVEFOLDER`
 - Default: Empty

Older SMAT Database history files can be saved outside the `SmathHistory` folder. The SMAT Database GUI provides an operation to copy/move the older history files to the `SmathHistory` folder.

Then, the files can be loaded into the SMAT Database GUI.

crontab SMAT Database cycling

crontab SMAT Database cycling is a method of cycling based on time. Cycling can be automatic and regular.

A `hcismatcycle.pl` perl script is located in `$HCIRoot/bin`. This can be invoked by a `crontab` task to cycle the `smatdb` base on time. You can make a `crontab` according to specific requirements. Instructions are provided on how to create and use a `crontab` script on UNIX and Windows.

The script contains eight arguments. Five of these are employed in SMAT Database cycling.

```
crontab [-s sitename] [-p processname] [-t threadname] [-b in|out] [-r HCIRoot]
```

The `crontab` script has these arguments for cycling:

- `-s [siteName]` is the site name.
- `-p [processName]` is the process name.
- `-t [threadName]` is the thread name.
- `-b in|out` is inbound or outbound.
- `[-r HCIRoot]` is the absolute path of a specified `HCIRoot`.

Only regular tasks about how to create `crontab` tasks are listed in these topics. Occasional timed tasks, such as `AT`, are not necessary.

For example, to invoke the perl script, you can write a new `hcismatdbcycle.bat` (Windows) or `hcismatdbcycle.sh` (UNIX).

After this, you can create one `crontab` task, and organize all cycling tasks within the `bat` or `shell` file.

Specific information is available for UNIX and Windows:

- [crontab in UNIX](#) on page 247
- [crontab in Windows](#) on page 249

crontab in UNIX

In UNIX:

- **Command:** `crontab`
- **Daemon:** `crond`
 - **Start Daemon:** `/etc/init.d/crond start | systemctl start crond.service`
 - **Stop Daemon:** `/etc/init.d/crond stop | systemctl stop crond.service`
- **Location:** `/var/spool/cron/etc/crontab`
- **Limitation:** `/etc/cron.allow && /etc/cron.deny ()`
Priority: `allow | deny`
- **Log:** `/var/log/cron`
- **Usage:** `crontab [-u username] [-l|-e|-r]`

Minute (0~59)	Hour (0~23)	Day (1~31)	Month (1~12)	Week (0~7)	Commands
*	*	*	*	*	

These characters can be used:

Character	Representation	Example
*	Run any time	0 15 * * 1 command
,	Run at sliced time	0 12 5,10,15 * * command
-	Run at scope time	30 9-18 * * * command
/n	Run every other time	*/10 * * * * command

Example for `crontab -e`:

Minute (0~59)	Hour (0~23)	Day (1~31)	Month (1~12)	Week (0~7)	Command
*/1	*	*	*	*	/bin/bash /the/location/of/hcismatdbcycle.sh

Example:

```
*/1 * * * *
```

This is a standard example of the `hcismatdbcycle.sh` file:

```
#!/bin/bash
setroot
hcismatcycle -s tsmatdbCycle -p smatdb2 -t conn_4 -b out -r /work/company/cis19.1/integrator
hcismatcycle -s tsmatdbCycle -p smatdb -t conn_2 -b out
...
```

UNIX example for smatdb cycle implementation

For this example, the site name is `test_site`. This site contains a process named `test_process` that has one thread named `test_thread`. This thread is set as **Save inbound messages** to inbound and **Save outbound messages** to outbound.

The embedded `crontab` in systems similar to Unix is used to finish the time-based `smatdb` cycle. This indicates you must have basic knowledge of `corn` and its time/date field setup. The `crontab` task is created to do the cycle on a regular basis.

To begin:

- 1 Run `setroot`, then start the engine and `test_process`.
- 2 To cycle `smatdb`:

```
$hcismatcycle -s test_site -p test_process -t test_thread -b in -r $HCIR00T/test_site
$hcismatcycle -s test_site -p test_process -t test_thread -b out -r $HCIR00T/test_site
```

- 3 You should maintain all cycle tasks in a special shell script. For example, create a shell file named `hcismatdbcycle.sh` and edit it:

```
#!/bin/bash
setroot
#task 1
hcismatcycle -s test_site -p test_process -t test_thread -b in -r /opt/cis19.1/integrator/
#task 2
hcismatcycle -s test_site -p test_process -t test_thread -b out
#task 3
hcismatcycle -s other_site -p other_process -t other_thread -b in -r /opt/cis19.2/integrator/
...
```

Note: If the `-r` option is not used, then the current `HCIRoot` is the default.

- 4 Create a crontab task of it:

```
$crontab -e
*/1 * * * * /bin/bash /the/path/of/hcismatdbcycle.sh
```

Set up a reasonable time schedule. In this example, the tasks in `hcismatdbcycle.sh` are run every minute.

On Linux, you can check the `cron` log at `/var/log/cron`. You can redirect the log to any specified location.

The scheduler task calls `cmd.exe` to run `hcismatdbcycle.bat`.

`setroot` requires administrator privilege. You can use PowerShell to ensure `setroot` was successfully set in the `bat` file. This is accomplished by running a command in `cmd.exe` similar to:

```
$schtasks /create /sc minute /mo 1 /tn "smatdbCycle" /tr "powershell -Command Start-Process -Verb RunAs C:\The\location\of\hcismatdbcycle.bat"
```

crontab in Windows

The Windows Task Scheduler schedules computer tasks to automatically run.

Unlike Linux, the Windows scheduler task does not provide an output log. Instead, you can redirect the log to a specified location of your choice.

The scheduler task calls `cmd.exe` to run `hcismatdbcycle.bat`.

The `setroot` command requires administrator privilege.

Example:

```
schtasks /create /sc minute /mo 1 /tn "smatdbCycle" /tr powershell -Command Start-Process -Verb RunAs C:\The\location\of\hcismatdbcycle.bat
```

Creating a task:

```
schtasks /Create
[/S system [/U username [/P [password]]]]
[/RU username [/RP [password]] /SC schedule [/MO modifier] [/D day]
[/M months] [/I idletime] /TN taskname /TR taskrun [/ST starttime]
[/RI interval] [ {/ET endtime | /DU duration} [/K]
[/XML xmlfile] [/V1]] [/SD startdate] [/ED enddate] [/IT] [/Z] [/F]
```

Deleting a task:

```
schtasks /delete
[/S system [/U username [/P [password]]]] [/TN taskname] [/F]
```

Deleting example:

```
schtasks /delete /tn "smatdbCycle" /f
```

Running a task:

```
schtasks /run
[/S system [/U username [/P [password]]]] [/TN taskname]
```

Running example:

```
schtasks /run /tn "smatdbCycle"
```

Ending a running task:

```
schtasks /end
[/S system [/U username [/P [password]]]] [/TN taskname]
```

Ending example:

```
schtasks /end /tn "smatdbCycle"
```

Querying for task Information:

```
schtasks /query
[/S system [/U username [/P [password]]]]
[/FO format [/XML] [/NH] [/V] [/TN taskname] [/?]
```

Querying example:

```
schtasks /query
```

Changing a task:

```
schtasks /change
[/S system [/U username [/P [password]]]] /TN taskname
{ [/RU runasuser] [/RP runaspassword] [/TR taskrun] [/ST starttime]
[/RI interval] [ {/ET endtime | /DU duration} [/K] ]
[/SD startdate] [/ED enddate] [/ENABLE | /DISABLE] [/IT] [/Z] }
```

hcismatdbcycle.bat example

This is a standard example of the hcismatdbcycle.bat file:

```
@echo off
REM COPYRIGHT 2019 INFOR. ALL RIGHTS RESERVED.
>>C:\the\real\path\task_smatdb.log 2>&1 (
echo Cycle time: %date:~0,4%%date:~5,2%%date:~8,2%%time:~0,2%%time:~3,2%%time:~6,2%
```



```
call setroot
call hcismatcycle -s tsmatdbCycle -p smatdb2 -t conn_4 -b out -r C:\company\cis19.1\integrator
call hcismatcycle -s tsmatdbCycle -p smatdb -t conn_2 -b out
...
)
```

The log is:

```
Cycle time: 20171228145130
HCIR00T path is C:\company\cis19.1\integrator
No default site -- no site set
Smatdb cycle now.
site name: 'tsmatdbCycle'
process name: 'smatdb2'
thread name: 'conn_4'
in/out: 'out'...
Process command port not defined!
Process is probably not running.
HCIR00T path is C:\cloverleaf\cis6.2\integrator
Smatdb cycle now.
site name: 'tsmatdbCycle'
process name: 'smatdb'
thread name: 'conn_2'
in/out: 'out'...
Response:
outbound message saving cycled
```

Windows example for smatdb cycle implementation

For this example, the site name is `test_site`. This site contains a process named `test_process` that has one thread named `test_thread`. This thread is set as **Save inbound messages** to inbound and **Save outbound messages** to outbound.

In Windows, the embedded task scheduler is used to finish the time-based smatdb cycle. To do this, you must have a basic knowledge of task scheduler setup.

You should maintain all task in a special bat file.

To do this, create a bat file named `hcismatdbcycle.bat` and edit it. For example:

```
@echo off
REM COPYRIGHT 2019 INFOR. ALL RIGHTS RESERVED.
>>C:\the\real\path\task_smatdb.log 2>&1 (
echo Cycle time: %date:~0,4%%date:~5,2%%date:~8,2%%time:~0,2%%time:~3,2%%time:~6,2%
call setroot
call hcismatcycle -s t -p smatdb2 -t conn_4 -b out -r C:\cloverleaf\cis19.1\integrator
call hcismatcycle -s tsmatdbCycle -p smatdb -t conn_2 -b out
...
)
```

On Windows, the scheduler task does not provide an output log. Instead, you can redirect the log to a location of your choice.

The scheduler task calls `cmd.exe` to run `hcismatdbcycle.bat`.

`setroot` requires administrator privilege. You can use PowerShell to ensure `setroot` was successfully set in the bat file. This is accomplished by running a command in `cmd.exe` similar to:

```
$schtasks /create /sc minute /mo 1 /tn "smatdbCycle" /tr "powershell -Command Start-Process -
Verb
RunAs C:\The\location\of\hcismatdbcycle.bat"
```

Recovering corrupted SMAT database data

The SMAT databases could get corrupted because of a Sqlite database locking control issue.

For example, when the engine and GUI simultaneously query the SMAT database, the IDE queries the SMAT database when the engine is cycling the SMAT database. In these cases, data loss could happen due to SMAT database corruption.

If this happens, then you can recover the SMAT database:

- 1 Dump the corrupted SMAT database into a text file.
- 2 Remove all error blocks from the text.
- 3 Add statements to create and populate the `smat_info` table if it was not created in the previous raw SQL statements.
- 4 Add `commit` statements (or `END TRANSACTION`) to the end of the dump text.
- 5 Change the raw dump text file to the `.sql` format.
- 6 Use the `sqlite3` command to read the `.sql` file to recreate the SQLite database and rename it to the `.smatdb` format.

Search results

Click **Resend Marked Messages** to resend all marked messages. Messages are marked by selecting the check box. The mark for each message is kept until you perform a new search or close the tool. The number of marked messages is shown at the bottom of the pane.

Click **Mark/Unmark All Messages** to select/clear all messages in the message list. You can also select/clear the check box on the table header to select/clear all of the messages on the current page. The **Show All** menu indicates whether the table needs to **Show All** messages, marked and unmarked, or **Show Marked** only messages. The navigation buttons assist you in navigating the search results, and are only enabled when the message list has multiple pages.

Message table

The message table only permits a single selection. When a message is selected, the message metadata and content are shown on the Metadata and Content tabs.

Sort messages by clicking the table headers. All headers can be sorted, except for Field/Alias Match. Modified messages have an asterisk (*) after the Message ID.

Sorting results are according to the original message's metadata, not modified messages.

The maximum number of messages a page can display is configurable within the message list. At the bottom of the table a list shows the maximum number of rows in each page. Choices are 50, 100, and 200. By default, the maximum number is 50. If this number is changed, then the maximum number is saved into `client.ini`. When you open the tool again, the saved maximum number is applied to the message list.

Customized message list

The displayed order and columns of the message list can be customized through the menu that opens when you right-click a column header. This menu lists the most often used columns. Check marks indicate whether the columns are currently shown on the list. The order of the items is the same as the display order of the columns in the table.

The **Customize** menu item opens the **Customize Columns** dialog box that lists all available metadata fields plus **File Name** and **File Path**. Select the fields on the list to specify which fields are shown on the message list.

Deleting SMAT Database messages

This feature is available in all security modes. A permission check is made in Advanced security mode.

All delete actions are written to the Audit Log.

The Search Results **Delete** button is located next to **Resend**, and is enabled when messages are selected.

To delete messages:

- 1 On the SMAT Database, select files, specify the search conditions, and click **Search**.
- 2 From the search results, select the messages to be deleted.

To select messages, you can:

- Select the messages to delete.
- or
- Select all messages and de-select the messages to not be deleted.

- 3 Click **Delete**.

When the operation has completed, a dialog box opens with the successful message deletion result or a failed deletion result.

Configuring the display message content size

In `$user.home/clide/client.ini`, the `msg_content_chunk_size` key controls the display message content size in one page.

To configure this:

- 1 Go to `$user.home/clide/client.ini` and add `msg_content_chunk_size=1024` to the `[general]` section. For example:

```
[general]
msg_content_chunk_size=1024
```

- 2 Restart the IDE. The message content displays in one page if the size is less than 1 MB.

Message metadata

When a message is selected in the message table, the Message Data panel shows the message data. This panel supports editing and non-editing modes.

This table shows the toolbar options:

Option	Description
Resend Current Message	Click to open the Resend Message dialog box, where you can resend the currently-selected message. SMAT database uses a similar Resend Message dialog box as in the SMAT tool. If the dialog box is to resend multiple messages, then fields such as context and priority are populated with the first message of the message set.
Apply	Click to commit the modifications. This is enabled only when the data panel is in editing mode and there are changes on a message. A reminder is displayed if you select another message or end the editing mode without committing the modifications. All modified messages are saved in memory and are purged after the tool instance is closed.
Enter/Exit Edit Mode	Switches between editing and non-editing mode. <ul style="list-style-type: none"> • In editing mode, you can modify the metadata and content of the displayed message. • In non- editing mode, you can only view the message data. • When a modified message is selected on the Message list, the Message Data panel is automatically in editing mode. In edit mode, Select Encode and Format is available for the Content tab. This opens the Configure Encode Scheme and Format dialog box for the currently selected message.

Metadata tab

This tab is a table that lists all metadata and their values in ascending order. The Name column displays the name of the metadata. The Value column displays the value of the metadata.

In editing mode, the value cells of editable metadata are also editable. When you select a value cell, the table shows a corresponding cell editor where you edit the value of the metadata. Non-editable metadata is gray.

Each metadata has other types of cell editors:

- The cell editors of Data Format, Group ID, Group Message ID, Protocol Driver Control, User Data, and User Defined Separators are text fields.
- The cell editors of External DB Table, Original Source Thread, Recovery Logging State, Source Thread, Skip Xlate, and Xlate Thread are boxes that list all available values for the metadata.
- The cell editors of Destination Threads and Original Destination Threads are a text field plus a button, because they permit multiple values. Using the editor, specify the value directly into the text field. You can also click the button to select threads from the **Select Threads** dialog box thread list.

- The message Flags metadata uses the same cell editor as Destination Thread. The text field is disabled as the value of the metadata is a hex number. The button opens the **Configure Flags** dialog box.

Content tab

In editing mode (by clicking **Enter Edit Mode**), this tab has two text areas.

- The left side displays the message content formatted with the specified encode format.
- The right side displays raw content text for the user to modify. In non-editing mode, the right side text area is hidden.

If the current search includes regex criteria for message content, then the left side text area highlights the content texts that match the regex criteria.

Note: When working with regex criteria, brackets are treated as characters. They do not form a capturing regex.

In editing mode, the text area that displays the formatted message content is not refreshed if there are any modifications on the raw content. You must first commit the modifications by clicking **Apply**.

The **Auto Wrap** check box is used to keep the content editor backward compatible. In previous versions, the message content editor always auto-wrapped the raw data. Now, this tab does not auto wrap the message content by default, but provides the check box to keep the original behavior.

The carrier return and line feed characters contained in the message raw data are replaced with the literal <CR> and <LF> characters when they are displayed in the text box. This replacement protects you from removing these particular characters when modifying the message data. The <CR> and <LF> characters are reverted back to the real carrier return and line feed when saving the modifications into the message object.

Configure Encode Scheme and Format dialog box

When in Edit mode, clicking **Select Encode and Format** opens the **Configure Encode Scheme and Format** dialog box for the currently selected message.

The **Encode Scheme** and **Encode Format** values are specific to the SMAT database to which the current message belongs.

The **Encode Scheme** menu lists all system-supported encode schemes.

The settings are retrieved from the corresponding `ecd` file:

- If the `ecd` file exists, then the dialog box is populated.
- If there is no `ecd` file, then the default encode scheme is ASCII no encoding is performed and the default format is None.

The **Encode Format** menu lists all available message formats.

Changes are saved into the `ecd` file.

For **Connection**, select the database connection from the list.

For **Table Schema**, select a schema from the list of all currently imported schemas under the database connection.

For **Detail level**, click the arrow to open a list of detail levels for the output that is reported in the text area. Detail levels go from 0 (raw, unparsed data) to 4 (most detail).

SMAT encryption

SMAT provides the option to enable encryption, and is also available in Global Monitor.

When set, SMAT data is stored in an embedded SQLite database in encrypted form. This is optional.

When not set, SMAT data is stored in the flat file format. This is for backward compatibility.

The SMAT database file name uses the `.smatdb` suffix. Cycled files use `TIMESTAMP.smatdb` or `.old.smatdb`, depending on the SMAT history configuration.

Note: Access to a particular `.smatdb` must originate from the same machine; otherwise, database corruption could occur. To prevent this, verify that `.smatdb` is not concurrently being accessed from another machine.

SMAT search

In the IDE, an integrated tool supports the database-based SMAT. This tool implements a cross-database search according to the user's requirement.

In Global Monitor, the SMAT search supports the two types of SMAT.

Database features include:

- Searching database messages across multiple SMAT databases
- Searching database messages by message content and most metadata
- Modifying and resending database messages
- Saving, editing, and restoring search criteria (IDE only)

Message resend: `resend_db` command

The `resend_db` engine command is for resending messages. This command reads messages from a temporary, and optionally encrypted, database and sends them forward.

The IDE and Global Monitor use `resend_db` when using a database.

The `resend` command is used for file-based SMAT.

The `-E` and `-D` commands are for the database option. The other options are the same as the `resend` command.

Note: "Base site" is the site on which the newly created site is based. For example, for `hcisiteinit.pl` this is the site name passed with the `-c` option, or `siteProto` without the `-c` option. For `hcirootcopy.pl`, it refers to individual sites in the site list that are passed by the `-s` or `-l` option.

Usage:

```
resend_db ib_pre_tps|ib_post_tps|ob_pre_tps|ob_post_tps filename
[-e msg_encoding] [-E [0|1]] [-D [sqlite]]
```

Option	Scope	Description
-E	<ul style="list-style-type: none"> + hcisiteinit.pl + hcicreatesite.pl + hcirrootcopy.pl + hcirsiterestore.pl * resend_db cmd 	<ul style="list-style-type: none"> • SMAT is encrypted (=1), or not encrypted (=0). • Encryption key is read from \$HCISITE • • + Indicates that values are stored in SMAT encrypted configuration in siteInfo of the new site, essentially overriding the value in base site. • * Indicates whether input files are encrypted. Default is 0.
-D	<ul style="list-style-type: none"> + hcisiteinit.pl + hcicreatesite.pl + hcirrootcopy.pl + hcirsiterestore.pl * resend_db cmd 	<p>The driver is used to read/write SMAT. Acceptable values are:</p> <ul style="list-style-type: none"> • file: SMAT is stored in .msg and .idx files. • sqlite: SMAT is stored in SQLite database (with .smatdb file extension). <p>Using -E=1 and -D=file is an invalid combination.</p> <p>+ indicates that values are stored in SMAT driver configuration in siteInfo of the new site, essentially overriding the value in base site.</p> <p>* indicates the driver to be used for input files. For resend_db, -D is implied.</p>

Note: SQLite is integral to engine functionality. When the first connection to a smatdb file is opened, wal and shm files are temporarily created. These are located in the same directory as the smatdb file and use the same name as the smatdb file. These are normally removed when the last connection to the smatdb file is closed. If you move the smatdb file, then you must also move these temporary files; otherwise, data is lost.

There are two formats for file based SMAT: old and new. This script converts between the file-based SMAT format and database-based SMAT format. This script can only be used as a database export utility.

When a SMAT database file is encrypted, the encryption information is read from \$HCISITEDIR/siteInfo environment variable or it can be overridden with the -s option.

hcismatconvert

There are two formats for file based SMAT: old and new. This script converts between the file-based SMAT format and database-based SMAT format. This script can only be used as a database export utility.

When a SMAT database file is encrypted, the encryption information is read from \$HCISITEDIR/siteInfo environment variable or it can be overridden with the -s option.

Note: The default is a forward conversion that overwrites the input file. If the input file name has the `.smatdb` extension, then this is a SMAT database export.

```
hcismatconvert [-b] [-f] input filename [-o output filename] [-s siteInfo path]
```

- `-b` is a backwards conversion, new to old. This does not apply to SMAT database (`.smatdb`) import/export.
- `-f` forces a database update. This is only for a database import.
If the output file exists, then the database import fails.
If a force is specified, then the data is imported to the existing database of that name.
- `input filename` is the input file named `filename`.
- `-o output filename` outputs to the file named `filename` instead of overwriting input. If the file name has the `.smatdb` extension, then this is a SMAT database import.
- `-s siteInfo path` is the siteInfo path for the SMAT database import/export.

Caution: Do not use this option for a file database export.

Example: Converting a file-based SMAT history to database-based SMAT

To do this, use a file-based SMAT as the input file and specify a database-based SMAT file as the output file.

For example, to convert the file-based SMAT files `file.idx`, `file.msg`, and `file.ecd`, use:

```
hcismatconvert file -o dbConverted.smatdb
```

This command locates the files with rootname "file" and then finds the `file.idx`, `file.msg`, and `file.ecd` files.

For the output, it creates `dbConverted.smatdb`, and `dbConverted.ecd`.

SMAT database encryption key customization

The SMAT database encryption key (SMATDB key), can be customized. This key is stored in `siteInfo` in the format `smatdbkey=XXX`.

It can be customized/modified when creating a site or changing the SMATDB key from the GUI. The `hcismatcrypt` tool is provided to validate/change the encryption key of a SMAT DB file.

Generally, all encrypted SMATDB files under a site share the same `smatdbkey` in `siteInfo`. If one SMAT database file does not have the same `smatdbkey`, then the GUI requires specifying the correct key when searching it.

See [Security server Encryption tab](#).

Creating a site with a specified SMATDB key

There are various methods to create a site:

- In the **Site Init** dialog box, **Encrypt SMAT databases** is enabled when **Save SMAT into databases** is selected. When this is selected, SMAT messages are encrypted when saved to the database. Specify a SMAT database password in the field.

The **Site Init** dialog box is located at **Start > All Programs > Infor Cloverleaf Integration Suite > Tools**.

- `hcicreatesite [-c site] [-p path] [-E flag]
[-D type] [-K key] root newsite`
- `hcisiteinit [-c site] [-s] [-i] [-p path] [-E flag]
[-D type] [-K key] newsite`

key is the SMAT database encryption key.

Changing the SMATDB key from the GUI

- 1 On the **Site Preferences > SMAT** tab, click **Change Password** to open the **Change Password** wizard. This lists the `smatdb` file list in the current site.
- 2 Click **Next** to change the password.
- 3 Specify the old password and new password and click **Apply**. This changes the password for all of the site `smatdb` files which match the specified old password. The `smatdb` files which do not match the specified old password are ignored.

This also changes the `smatdbkey` of `siteInfo`, and applies the new `smatdbkey` to SMATDB search.

Logic of loading the SMATDB key

When the SMATDB file is encrypted, the SMATDB key in `siteInfo` is decrypted and is used to access the file.

- If SMATDB key is empty or the key does not exist, then the site name is used as the password for access.
- If an encrypted database file cannot be accessed, then a **Password Prompt** dialog box opens for specifying the correct password.
- If an encrypted database cannot be accessed, then its messages are not shown in the GUI.

When the SMATDB file is un-encrypted, it can be directly accessed.

Migration from pre-6.2 to later versions

During migration from versions earlier than 6.2 to later versions, `hcirootcopy` sets the site name as SMATDB key.

hcismatcrypt tool usage

```
hcismatcrypt command [args]
```

command is one of:

- `validatekey` validates the encryption key.
- `rekey` changes the encryption key.

Use `hcismatcrypt command -H` for usage of a specific command.

SQLite error log

In situations where there is corruption in the SMAT database, there is a SQLite error log to help in debugging.

SQLite can be configured to invoke a callback function containing an error code and a terse error message whenever anomalies occur. This mechanism is helpful in tracking obscure problems that occur rarely and in the field.

If there are errors when the engine writes messages into the SMAT database, then the errors are recorded in the engine log.

When doing a SMAT database search, the errors are recorded in `$HCIR00T/server/logs/smat.log`.

The `smat.log` file is empty if you are using local mode (**Use the Local Config Server** is selected on the **Select an Infor Cloverleaf Integration Services Server and Environment** dialog box).

To connect to the host server, select **Use a Remote Config Server**.

See <https://www.sqlite.org/errlog.html>

Migrating SMAT database files

If you do not require migrating SMAT when you migrate your sites with `hcirrootcopy` but require access to the SMAT database files, then use `hcirrootcopy` to migrate only the SMAT database files.

When `hcirrootcopy` is not used for the SMAT migration, you can use `hcismatcrypt` to migrate SMAT.

By default, `hcirrootcopy` copies all the SMAT database files to the new site and then migrates them.

If option `-N` is specified, then it does not do the copy and migration. In this instance, use `hcismatcrypt` to manually migrate the SMAT database files.

Usage:

```
hcismatcrypt migrate [options]
```

Options are:

- `smatdbName/smatdbDir`:
This is the SMAT database name or directory.
If it is a directory, then all `*.smatdb` files under `Dir` are migrated.
- `smatdbKey`:

This is the encryption key.

- `hcismatcrypt -H:`

This lists all available commands.

Note: You can also send the CLAPI request using the `cURL` command.

For additional information, see:

- [hcirootcopy](#) on page 1371
- [hcismatcrypt](#) on page 1387

In CIS 19.1 and later, CL-API `hcismatcrypt` can be configured in allow list databases and run by `CL_API`. To do this, use these steps:

- 1 Ensure CLAPI and the CLAPI document are enabled. The administrator can verify this on the **ServerAdmin > Web Server** tab.
- 2 Locate the `smatdb` to migrate, and get the absolute file path. For example, `C:\cloverleaf\cis19.1\integrator\testmigrate\case1.smatdb`.
- 3 If the CLAPI document is enabled, then open the swagger-UI page using `https://hostname:15047/clapi/swagger-ui.html`.
- 4 Locate `misc-command-controller`, and select `POST /api/site/{sitename}/command/execute`. Use these query parameters:

- `sitename:`
`smatdb` file. For example, This is the name of the site that stores the `testmigrate`.
- `command:`
This is `hcismatcrypt migrate smatdb file path`.

For example:

```
hcismatcrypt migrate C:\cloverleaf\cis19.1\integrator\testxlt\case1.smatdb smatdb password
```

`password` is the password for the `smatdb` file.

- `timeout` is the maximum time (ms) for the migration. For example, `50000`.
- 5 Click **Try it out**. You should get a "200" response code. This indicates success, and that there is no error, or any other, message in the response.

Note: You can also send the CLAPI request using the `cURL` command.

Migrating encrypted SMAT databases

In some instances, you must create an encrypted SMAT database on another server. For example, if the primary server is dead, you can still use the encrypted SMAT database on a backup server.

To do this and ensure the SMAT database can be decrypted on the secondary server:

- 1 Create an encrypted SMAT database on server A.
- 2 Close all connections before moving files.

- 3 Move these database files to server B, using the same site name and directory path as the original: `dbName.smatdb`, `dbName.smatdb-wal`, `dbName.smatdb-shm`, `dbName.smatdb-sec`, `dbName.smatdb-sec-wal`, `dbName.smatdb-sec-shm`

Some files might not be present. You must check the consistency of `dbName.smatdb` and `dbName.smatdb-sec`.

The database can now be used at the new server. The site name must be the same as the original.

Issues with renamed SMAT database sites

The site name is used as a key to encrypt and decrypt the SMAT database. An error can happen when a SMAT database site is renamed and an attempt is made to access archived SMAT DB files.

Messages in the SMAT DB files become inaccessible from the time the site was renamed.

For example:

- 1 A site named `siteA` is renamed to `siteB`.
- 2 In `siteA`, the SMAT database is encrypted with key A.
- 3 After renaming the site, the SMAT database that was encrypted with key A cannot be accessed with key B. This is the key from `siteB`'s SMAT database.
- 4 `xxx.smatdb-sec` is generated to store SMAT messages. You cannot find any messages in the SMAT database files since the time the site was renamed.

Changing the key to the current site name

The site name is the only factor in the decrypting process. Data is inaccessible unless you change the key to the current site name in a shell. To change the key in a shell, run these commands in this order:

- 1 `sqlite3 XXX.smatdb`
- 2 `PRAGMA KEY = 'wrong key (2nd site name?)'`
- 3 `PRAGMA REKEY = 'correct key (1st site name?)'`
- 4 `.quit`

Recovering the data

To recover the data, for each `.smatdb` file (including `-sec`, but not `-wal`), identify which key is used for which site by running these commands:

- 1 `sqlite3 this.smatdb`
- 2 `PRAGMA KEY = 'first site name'`
- 3 `Select count (*) on smat_msgs`
- 4 `.quit`

If the above steps work, then you can stop now. If not, then continue with these steps:

- 5 `sqlite3 this.smatdb`

```
6 PRAGMA KEY = 'second site name'
7 Select count (*) on smat_msgs
8 .quit
```

Changing an incorrect key

To correct an incorrect key in `.smatdb`, run these commands:

```
1 sqlite3 this.smatdb
2 PRAGMA KEY = 'wrong key (2nd site name?)'
3 PRAGMA REKEY = 'correct key (1st site name?)'
4 .quit
```

sqlite command

Using the `sqlite` command is similar to `hcismat`, which is used to access old SMAT data. The `sqlite` command is used for accessing the SMAT database.

Accessing SMAT database using sqlite command

To access a SMAT database using `sqlite`:

```
1 sqlite3 <smatdbName>
2 PRAGMA KEY = <siteName>
   Ignore this step if the SMAT database is not encrypted.
3 <sql to query>
   For example, select * from smat_msgs.
4 .quit
```

Accessing SMAT database using Tcl script

To access a SMAT database using a Tcl script:

```
1 package require sqlite
2 sqlite3 db smatdbName
3 db eval "PRAGMA KEY=<siteName>"
   Ignore this step if the SMAT database is not encrypted.
4 db eval "SELECT load_extension('sqlite3-regex.dll')"
   Load sqlite3-regex.so on UNIX.
5 db eval "sql to query"
6 exit
```

SQL extension support

CONFREGEXP: Specifies the encoding of message content.

Usage:

```
CONFREGEXP('ECD', 'dbalias1=encoding1;dbalias2=encoding2')
```

SMATREGEXP: Queries message content that match given `regex`.

Usage:

```
SMATREGEXP(regex, rowid, dbalias)
```

REGEXP: Queries columns of a message that match given `regex`.

Usage:

```
SMATREGEXP(regex, columnName)
```

SMATREGEXPDETAIL: Gets the position offset of matched `regex`.

Usage:

```
SMATREGEXPDETAIL(regex, rowid, dbalias)
```

Examples

To configure the encoding of database `attach1` as `latin1`:

```
select CONFREGEXP('ECD', 'attach1=latine1')
```

To search all columns and highlights of HL7 messages, where the message type is `adt^a31` and to have it ordered by message ID:

```
select *, SMATREGEXPDETAIL('(\\r)*?msh\\|([\\^\\|]*\\|){8}+[\\^\\|]*adt^a31', rowid, 'main') from smat_msgs where (SMATREGEXP('(\\r)*?msh\\|([\\^\\|]*\\|){8}+[\\^\\|]*adt^a31', rowid, 'main') >0 ) order by MidDomain asc, MidHub asc, MidNum asc
```

smat_msgs table columns

This table lists the contents of the `smat_msgs` table columns:

Column	Data type
1 MessageContent	BLOB
2 DataLen	INTEGER
3 DestConn	VARCHAR
4 OrigDestConn	VARCHAR

Column	Data type
5 DataFmt	VARCHAR
6 SepChars	VARCHAR
7 DriverCtl	VARCHAR
8 UserData	VARCHAR
9 MidDomain	INTEGER
10 MidHub	INTEGER
11 MidNum	INTEGER
12 SrcMidDomain	INTEGER
13 SrcMidHub	INTEGER
14 SrcMidNum	INTEGER
15 Type	VARCHAR
16 SourceConn	VARCHAR
17 OrigSourceConn	VARCHAR
18 Time	INTEGER
19 Priority	INTEGER
20 SaveContext	VARCHAR
21 Flags	INTEGER
22 State	INTEGER
23 GroupMidDomain	INTEGER
24 GroupMidHub	INTEGER
25 GroupMidNum	INTEGER
26 XltThread	VARCHAR
27 Retries	INTEGER
28 SkipXlt	INTEGER
29 UseRecoverDB	INTEGER
30 GroupID	INTEGER
31 TimeIn	INTEGER
32 TimeXlt	INTEGER
33 TimeOut	INTEGER

Column	Data type
34 TimeQCur	INTEGER
35 TimeQTot	INTEGER
36 TimeRec	INTEGER
37 TimeStor	INTEGER
38 TimeArc	INTEGER
39 DBTable	VARCHAR
40 Flags_Resent	INTEGER
41 Flags_Trace	INTEGER
42 ErrorString	VARCHAR

Deleting old messages

In the SMAT database, there are no columns that directly show a message's age. You can use the Time or TimeIn columns to calculate the age.

These columns are INTEGER type in the `smat_msgs` table.

- 1 Access the SMAT database.
See [Accessing SMAT database using sqlite command](#).
- 2 Use the `sqlite` command:

```
delete * from smat_msgs where condition
```

For example, the current time is 1452510267000 milliseconds, and the time older than 10s is 1452510257000.

In this case, messages older than 10s can be deleted using:

```
delete * from smat_msgs where time <= 1452510257000
```

SMAT HL7 smart search

The SMAT HL7 Smart Search feature is provided in the SMAT database tool. You can search messages without knowledge of the underlying HL7/HPRIM data formats.

The SMAT HL7 Smart Search feature builds regular expressions from information provided by the user. The user enters HL7/HPRIM field descriptions such as patient name, or field locations such as PID.5, along with the search criteria. The default search is case-insensitive.

The search builds the regular expressions, searches SMAT, and returns the results. These regular expressions can be saved for future use.

You can search using two query types:

- Abstract Queries
- Logical Queries

These are translated into corresponding regular expression queries and run using regular expressions.

Queries can be arbitrarily combined with logic operators (for example, `and` or `or`) and grouped with parentheses to make up an advanced query.

Additional features include:

- Subfields use 1-based numbering.
- The **MSH** field starts with MSH.2 for the first field.

HL7/HPRIM separator cache

The separators that are used for the SMAT search are retrieved from the first message of the searched `smatdb` file. These separators are cached during the first search.

Default separators are used if no separators are retrieved from the first message.

If the first message begin with MSH/H and the first two field separators are the same, then the separators are retrieved in sequence. Otherwise, default separators are used.

Search options

Text search

The SMAT GUI contains HL7 and HPRIM search options for a text-based search and an address search. The default search is case-insensitive.

Example:

```
"Patient Name" = "Doe"
```

Address search

The address search can use `segment.field#` notation or a field description.

Field descriptions are matched from the HL7 fields database.

You can use substrings. For example, `order` would match `placer order number`, `filler order number`, and any other field name containing `order`.

The address is case-insensitive.

You can search subfields for more direct matches.

Aliases

An alias is created to associate the `segment.field` number to a common name. For example:

- PID.5.1 = Lastname
- PID.3.1 = MRN
- ORC.2.1 = Accession Number

A list of standard aliases is provided. You can also create and save custom aliases. This is configured on the **Site Preferences > Aliases** tab.

Variant search

Variant formats can also be searched. This search uses the site first. If the variant is not found in the site, then the primary is then searched.

Basic queries

This table shows basic queries and their corresponding regular expressions:

Query	Code
<Filed Name>=value	<code>(\r)?+Segment\ ([^\]*\){Index-1}+[^\]*value</code>
Not <Segment>.<Index>=value. This indicates the carriage return displays once or not at all before a segment.	<code>indicates the carriage return could(\r)?+Segment\ ([^\]*\){Index-1}+(?!value)</code>
<Segment>.<Index>=value	<code>(\r)?+Segment\ ([^\]*\){Index-1}+[^\]*value</code>
<Field Alias>=value	<code>(\r)?+Segment\ ([^\]*\){Index-1}+[^\]*value</code>
<Subfield Alias>=value	<code>(\r)?+Segment\ ([^\]*\){Index-1}+([^\ ^]*\^){SubIndex-1}+value</code>

In these queries:

- `(\r)?+Segment\|` matches the specified segments in a message.
- `(\r)?+`
- `([^\|]*\|){Index-1}+` specifies how many fields should be passed through before reaching the specified field.
- `Index` indicates the field index.
- `[^\|]*value` attempts to match a field value. A field consists of several subfields. To support wildcard matching, `[^\|]*` is added before the value..
- `(?!value)` introduces a zero-width negative lookahead to represent the logic NOT.
- `([^\|^]*\^){SubIndex-1}+value` matches the specified subfield.

- `([^\^]*\\^){SubIndex-1}+` specifies how many subfields should be gone through before reaching that subfield.
- The `|` and `^` characters are the default delimiters of fields and subfields. They are replaced with the real delimiters in runtime if the messages are using the other delimiters.

Advanced queries

Basic queries can be joined by logical operators and grouped by parentheses to form an advanced query.

The logic AND is implemented using SQL's INTERSECT and the logic OR is implemented with UNION ALL.

When basic queries in the advanced query are run, the result set is unioned or intersected using SQL's UNION ALL or INTERSECT separately. This is accomplished according to the operators of the basic queries.

These are the advanced queries and their corresponding SQL statements:

Query	Code
<Filed Name1>=value1 and <Filed Name2>=value2	<pre>select rowid from smat_msgs where smatregexp('regex1', rowid, 'database alias') > 0 INTERSECT select rowid from smat_msgs where smatregexp('regex2', rowid, 'database alias') > 0</pre>
<Filed Name1>=value1 and <Filed Name2>=value2	<pre>select rowid from smat_msgs where smatregexp('regex1', rowid, 'database alias') > 0 INTERSECT select rowid from smat_msgs where smatregexp('regex2', rowid, 'database alias') > 0</pre>
(<Filed Name1>=value1 or <Filed Name2>=value2) and <Filed Name3>=value2	<pre>Select rowid from (select rowid from smat_msgs where smatregexp('regex1', rowid, 'database alias') > 0 UNION ALL select rowid from smat_msgs where smatregexp('regex2', rowid, 'database alias') > 0) INTERSECT select rowid from smat_msgs where smatregexp('regex3', rowid, 'database alias') > 0</pre>
(<Filed Name1>=value1 and <Filed Name2>=value2) or <Filed Name3>=value2	<pre>Select rowid from (select rowid from smat_msgs where smatregexp('regex1', rowid, 'database alias') > 0 INTERSECT select rowid from smat_msgs where smatregexp('regex2', rowid, 'database alias') > 0) UNION ALL select rowid from smat_msgs where smatregexp('regex3', rowid, 'database alias') > 0</pre>

Use cases

These use cases show various search methods:

- Text search for messages for patient name
- Address search for ADT messages for patient name
- Text search for messages with order in HL7 field name

Text search for messages for patient name

An engineer has a request to find all messages for a patient name Doe.

The engineer uses the text search option specifying `patient name` and the text `stringstring` to search for Doe.

SMAT HL7 Smart Search finds all messages matching this patient name in any field name matching `patient name`.

Address search for ADT messages for patient name

An engineer a request to find all ADT messages for "PID.5.0 Doe" and "PID5.1 equals John".

The engineer uses an address search using `MSH.6.0 = ADT and PID.5.0 = Doe and PID.5.1 = John`.

SMAT HL7 Smart Search returns all ADT messages matching the PID selection criteria.

Text search for messages with order in HL7 field name

An engineer has a request to find all order and result messages containing "123456".

The engineer uses the text search option specifying `order` and text `123456`.

SMAT HL7 Smart Search returns all messages that match field names and placer names containing `order number`, `filler order number`, and others. Smart Search also returns the text matches for the field value `123456`.

Maintaining field aliases and field segment definitions

The field segment database and the aliases database on the site level stores user-customized format definitions and aliases. The **Site Preferences > Aliases** tab lists all existing aliases for the current site.

Add/edit/remove a site's field segment definition using the existing HL7/HPRIM tool.

For HL7 2.7 and HPRIM variants, changes on segments and fields are synced to the field segment database in the site.

Search query variations

Logical operators can be specified (for example, and, not, or) to connect two or more search queries.

You can also use grouping parentheses after the order of operations.

A valid query is composed of a field name or path name followed by an equal sign and a search string.

Examples

- Patient Name = Gavin and not Patient Name = Haller

- Patient Name = Gavin and Next of Kin Birth Place = 2643 Elm St., Michigan
- not Patient Name = Gavin
- (Patient Name = Gavin and Next of Kin Birth Place = 2643 Sparrow St. NY, NY) or (Patient Name = Gavin and Next of Kin Birth Place = 716 W 28th St. Los Angeles, CA)

In these examples, Abstract or Logical Queries are all text except the logical operators (in this example, and, not, or).

criteria_defusername.ini

Queries are saved in `criteria_defusername.ini`.

For example:

```
[def_name]
condition_0 = HL7 {{Patient Name[PID.5] = Irving and PID.3 = 200060}}
```

The corresponding field index is saved together with the field name or alias.

RegEx limitations

These regular expression (RegEx) limitations could be encountered when using the SMAT database:

- The circumflex "^" and dollar sign "\$" metacharacters could incorrectly match in the middle of the message if the target pattern matches at the start or end of a 1024-byte block.
- The pipe "|" metacharacter might not match correctly when the match happens at the end of a 1024-byte block.

For example, the pattern `1234|3789` does not match subject string `ABC123789` if `ABC123` ends at a 1024-byte block and `789` continues into the next byte block. This is because the character `3` happens in both sub-patterns `1234` and `3789` and the match is at a block boundary.

- Capturing parenthesis and back references (including backward assertions `\b` or `\B`) are not supported, and if used could result in an engine crash.
- `\r\n` can be used for searching new line on Windows, and `\n` can be used for UNIX.
- SMAT encodings that are not supported by the ICU library are treated as UTF-8 strings for regex.
- When the engine and IDE access the SMAT database simultaneously, this must be from the same system. Otherwise, database corruption could happen.

This scenario is possible when a `.smatdb` file is present on a network file system (NFS), is being written to by the engine, and is read from another host by the IDE.

Multiple IDEs accessing the same `.smatdb` file from several hosts is not an issue, unless the engine is writing to `.smatdb`.

Note: The encrypted SMAT database is tied to the site under which it was created. It is important to remember the site name for each `.smatdb` that was moved out of the site. For example, for archival purposes. All `.smatdb` files should be stored under the folder named by the site. If the site name is changed, then the `.smatdb` file created under the old site name does not work. Contact Support before undergoing such a scenario.

External message flags bit masks

For users who require a binary mapping of bit values to Message Flags, the External Message Flags Bit Masks table lists the externally visible bit values that are visible through Tcl.

For example, this is useful for those writing a Tcl script to convert the old format SMAT files that did not have metadata to the new style that has the metadata. `msgmetaset` does not work because there is no message handle. Knowing the flag names also does not work.

This table lists the binary position of each flag, so that you can construct the `FLAGS` field:

Bit	Usage
Bit 0x00000001	<ul style="list-style-type: none"> Tcl Flag Name: <code>should_be_freed</code> Used by internal memory management. Tcl Writable: FALSE
Bit 0x00000002	<ul style="list-style-type: none"> Whether the local copy of the message can be destroyed if sent to a thread in another process. Tcl Flag Name: <code>icl_owns_data</code> Tcl Writable: FALSE
Bit 0x00000004	<ul style="list-style-type: none"> Used by internal memory Automatically set when the thread is configured to await replies. When set, the threads await a reply after writing the message through the protocol connection. Tcl Flag Name: <code>expect_reply</code> Tcl Writable: TRUE
Bit 0x00000008	<ul style="list-style-type: none"> Set if the message is ever forwarded from one thread to another. Tcl Flag Name: <code>is_forwarded</code> Tcl Writable: FALSE
Bit 0x00000010	<ul style="list-style-type: none"> TRUE when MSG is on a queue; otherwise, FALSE. Tcl Flag Name: <code>is_enqueued</code> Tcl Writable: FALSE
Bit 0x00000020	<ul style="list-style-type: none"> Set to indicate that this message is the last in a related group. Tcl Flag Name: <code>last_in_group</code> Tcl Writable: TRUE
Bit 0x00000040	<ul style="list-style-type: none"> Set by the protocol driver to describe a protocol-send failure due to time-out. Intended for use by <code>SENDFAIL</code> TPS procedures. Tcl Flag Name: <code>proto_timeout</code> Tcl Writable: TRUE

Bit	Usage
Bit 0x00000080	<ul style="list-style-type: none"> Set by the protocol driver to describe a protocol-send failure due to nak. Intended for use by SE NDFAIL TPS procedures. Tcl Flag Name: proto_nak Tcl Writable: TRUE
Bit 0x00000200	<ul style="list-style-type: none"> Set if this message was resent back into the engine. Tcl Flag Name: is_resent Tcl Writable: TRUE
Bit 0x00000800	<ul style="list-style-type: none"> Recovers the message from the database. Tcl Flag Name: recovered Tcl Writable: TRUE
Bit 0x00001000	<ul style="list-style-type: none"> Tcl Flag Name: is_on_disk Tcl Writable: FALSE
Bit 0x00002000	<ul style="list-style-type: none"> Tcl Flag Name: keep_on_disk Tcl Writable: FALSE
Bit 0x00008000	<ul style="list-style-type: none"> Set to indicate that this message goes to the recovery database if there is any engine failure. Tcl Flag Name: use_rdb Tcl Writable: FALSE
Bit 0x00010000	<ul style="list-style-type: none"> Set if this message has a prewrite procedure. Tcl Flag Name: prewrite_done Tcl Writable: FALSE

SMAT database API

The SMAT database API is for storing messages in the SMAT database during message processing. You can manually store messages in the SMAT database at any user point of control (UPoC).

You can also use the API in UPoC. See [Tcl SMAT API](#).

These use cases show how to use the SMAT database API:

- Example: Pre/Post translate archive
User 1 must archive the current message before and after translation.
User 1 configures the steps in the Translation Configurator's **Pre Proc** and **Post Proc** tabs.
- Example: Conditional message archive
User 1 must keep copies of messages that meet a specific criteria. User 1 writes a Tcl proc that tests for the required conditions. If the message matches the conditions, then it is archived in the assigned SMAT database.

- Example: Archive decrypted CSC transactions

User 1 has written an interface that receives encrypted Cloverleaf Secure Courier (CSC) transactions. Before processing the CSC transactions, User 1 must archive the unencrypted transactions in the SMAT database.

User 1 writes a Tcl proc and loads it into the inbound TPS in NetConfig. When a transaction is received, it is archived in the SMAT database by the proc in the inbound TPS.

Searching SMAT Database messages with EBCDIC encoding

To search SMAT Database messages with EBCDIC encoding, use the SMAT API to save message content into a separate SMAT Database under the site. This is not inbound or outbound SMAT Database. This is accomplished with ASCII encoding by UPoC before converting to EBCDIC.

You can search this SMAT Database with ASCII through the GUI by opening the SMAT Database database-searching tools.

For example, in a Tcl proc:

```
# open a smatdb
set DBHandle [smatdbopen insert.smatdb]
# may need to convert the msg content from EBCDIC to ASCII if
# necessary
# set msg [convertMsgtoAscii $msg]
# insert a msg
smatdbinsert $DBHandle $msg
smatdbclose $DBHandle
```

Tcl SMAT API

Because of the audit role of SMAT, for inbound, users can see exactly what was received in the NetConfig configured SMAT archives. For outbound, users cannot. The message that is saved is the one before encoding was applied because it is resent to an OB pre/post SMS queue. Encoding back to UTF-8 to insert into that queue can introduce anomalies.

When encoding is involved, you can use the SMAT API to save the post-encoded messages and view exactly what is being sent.

A Tcl interface has been implemented to store those messages into the SMAT database.

The SMAT API accesses the traditional IB/OB SMAT database, which is defined on the GUI to script actions that are based on the content.

You can save inbound/outbound messages into the SMAT database using the traditional IB/OB SMAT database. The SMAT API can save messages with other TPS contexts into an additional user-defined SMAT database that is separate from the traditional IB/OB SMAT database.

The user-defined SMAT database has these features (compared with the traditional IB/OB SMAT database):

- Full operations

You cannot use the SMAT API to save messages into the IB/OB SMAT database defined on the GUI. The SMAT API can open a traditional IB/OB SMAT database, but is restricted to read-only operations. Full use is granted to the user-defined SMAT database.

- **Multiple writers**
One protocol thread can only have one traditional IB/OB SMAT database. It can only be written based on its configured thread with a unique save context, that is, inbound or outbound. Multiple user-defined SMAT databases can be created in a thread or in a TPS. Similarly, one user-defined SMAT database can be shared/operated at the site level, that is, it can have multiple writers with multiple contexts by multiple processes.
- **Resend mapping**
For the resend of a user-defined SMAT database, there is a mapping between the TPS context (saveContext), and the traditional IB/OB save contexts.
- **Trace storage**
Trace information, such as the SMAT database name/location, is stored into the tracing database.

These behaviors are shared between the two types of SMAT database:

- Table schema is the same.
- The SMAT history and cycle logic share the same configuration with the traditional IB/OB SMAT database.
- The encryption key is the same as the SMAT database, that is, the site name.

Additional points:

- Tcl and java TPS are supported for the SMAT API.
- External Tcl scripts, for example, Tcl in an `hcitcl` shell, are also supported.
- The Java TPS is still through Tcl, not natively.
- `xltp` and `jxltp` are not supported.

Tcl API

This table describes the Tcl API commands:

Command	Description	Parameters	Returns
smatdbopen	Creates or opens a SMAT database named <code>SmatdbName</code> .	<code>smatdbName</code> <code>[-e msgEncoding]</code> <ul style="list-style-type: none"><code>smatdbName</code>: SMAT database name<code>msgEncoding</code>: Message content encoding	<p>SMAT database handle</p> <p><code>matdbName</code> can be an absolute or relative path. This is consistent with NetConfig.</p> <p>A duplicate <code>smatdbName</code> can be opened at the TPS of any thread or process of a site. This gives the option to save all into one, for example, saving all the messages into one database in the <code>exec</code> directory.</p> <p>If <code>-e msgEncoding</code> is specified, then the API uses this encoding to open <code>smatdb</code> and search the message content. If not specified, then it uses UTF-8 as the default, or uses the one in NetConfig that is based on the context.</p>

Command	Description	Parameters	Returns
smatdbinsert	Stores a message into the SMAT database.	<p>smatdbHandle msgID ?saveContext?</p> <ul style="list-style-type: none"> • smatdbHandle: SMAT database handle • msgID: Message object handle • saveContext: User-specified context. This is optional. The default is tps context. 	<p>Error code. 0 indicates insert success; otherwise, failure.</p> <p>saveContext is similar to a tag that can be used to identify the type/context of data being saved. It can be used for resending mapping and is saved into the SMAT database as metadata saveContext.</p> <p>It cannot be inbound or outbound. If inbound or outbound is used, then the default tps context is used.</p> <p>For the default tps context, see Default saveContext for each TPS.</p> <p>For external script, default context is empty.</p> <p>When an error code is returned, there are two ways to find the error detail. You can view the engine log, or locate detail according to the error code, since the return code stays consistent with the sqlite error return.</p> <p>If attempting to insert a message into the traditional IB/OB SMAT database, then an error is reported.</p> <p>If tracing is supported, then this function also inserts the record there.</p>

Command	Description	Parameters	Returns
smatdbcount	Gets the message total count that meets the search condition.	smatdbHandle ?searchCondition? smatdbHandle: The SMAT database handle. searchCondition: The SQL where condition for doing the search.	Message count. 1 means failure. If searchCondition is not specified, then this returns the total count of messages in the SMAT database.
smatdbsearch	Gets messages from the SMAT database.	smatdbHandle ?searchCondition? ?columnBitMask? smatdbHandle: The SMAT database handle searchCondition: The SQL where condition for doing the search. columnBitMask: The columns to search. The default is all columns. This must start with 0x.	Message object handle list. If no search-related parameters are specified, then this returns all the messages in the SMAT database. The table name of the SMAT database is smat_msgs. For SMATDB column definitions, see Table smat_msgs definition . For information on how to define columnBitMask, see SMAT database columns . If a column is not searched, then its value is empty.
smatdbdelete	Deletes messages from the SMAT database.	smatdbHandle ?deleteCondition? smatdbHandle: SMAT database handle deleteCondition: The SQL where condition for doing the delete.	Message number deleted. -1 means failure. If no optional parameters are specified, then all the messages are deleted in the SMAT database. If attempting to delete a message from the traditional IB/OB SMAT database, then an error is reported.
smatdbcycle	Cycles the SMAT database.	smatdbHandle smatdbHandle: SMAT database DB handle	0 indicates cycle success; otherwise, failure.

Command	Description	Parameters	Returns
smatdbclose	Closes a SMAT database.	smatdbHandle smatdbHandle: SMAT database handle	Error code. 0 indicates close success; otherwise, failure. The SMAT database handle must be removed when the SMAT database is closed.

DispositionList

This has been extended to support saving the user-defined `errContext` into the Cloverleaf error database.

```
lappend dispList "ERROR {$mh $errorContext}"
```

After returning the `dispositionList`, the message with `$mh` is saved into the error database. `$errorContext` is appended to the current `errContext` column of the error database.

Java UPoC API

```
public class SMATDB {
    /**
     * SMAT DB constructor
     * @param java upoc clover ENV
     * @param SMAT DB name
     * @return Error code. 0 Means insert success. Not 0 means failure.
     */
    public SMATDB(CloverEnv xCloverEnv, String SMATDBName){
    }
    /**
     * SMAT DB constructor
     * @param java upoc clover ENV
     * @param SMAT DB name
     * @param SMAT DB message content encoding
     * @return Error code. 0 Means insert success. Not 0 means failure.
     */
    public SMATDB(CloverEnv xCloverEnv, String SMATDBName, String encoding){
    }
    /**
     * open a SMAT DB
     * @return Error code. 0 Means insert success. Not 0 means failure.
     */
    public int open() {
    }
    /**
     * insert a message into SMAT DB
     * @param message to insert. The saveContext is tps context.
     * @return Error code. 0 Means insert success. Not 0 means failure.
     */
    public int insert(Message msg) {
    }
    /**
     * insert a message into SMAT DB with specified saveContext
     * @param message to insert
     * @param saveContext user-specified saveContext
     * @return Error code. 0 Means insert success. Not 0 means failure.
     */
}
```

```

    */
    public int insert(Message msg, String saveContext) {
    }
    /**
     * get total message count of smat db
     * @return message number. -1 means failure.
     */
    public int count() {
    }
    /**
     * get message count that meets condition
     * @param searchCondition sql where condition
     * @return message number. -1 means failure.
     */
    public int count(String searchCondition) {
    }
    /**
     * search all messages from SMAT DB
     * @return Error code. 0 Means insert success. Not 0 means failure.
     */
    public Message[] search() {
    }
    /**
     * search SMAT DB
     * @param searchCondition sql where conditions is supported
     * @return Error code. 0 Means insert success. Not 0 means failure.
     */
    public Message[] search(String searchCondition) {
    }
    /**
     * search SMAT DB
     * @param searchCondition sql where conditions are supported
     * @param columnBitMask
     * @return Error code. 0 Means insert success. Not 0 means failure.
     */
    public Message[] search(String searchCondition, String columnBitMask) {
    }
    /**
     * delete all messages from SMAT DB
     * @return Error code. 0 Means delete success. Not 0 means failure.
     */
    public int delete() {
    }
    /**
     * delete messages from SMAT DB
     * @param deleteCondition sql where condition is supported
     * @return Error code. 0 Means delete success. Not 0 means failure.
     */
    public int delete(String deleteCondition) {
    }
    /**
     * cycle SMAT DB
     * @return Error code. 0 Means insert success. Not 0 means failure.
     */
    public int cycle() {
    }
    /**
     * close SMAT DB
     * @return Error code. 0 Means insert success. Not 0 means failure.
     */
    public int close() {
    }
}

```

An add function with three parameters has been added to class `DispositionList` to save `errorContext` into the Cloverleaf error database:

```

public void add(int disposition, Message msg, String errorContext) {
}

```

Standard Tcl proc

An `hciSMATInsert` root level standard Tcl procedure is provided.

Parameters: `{smatdbname XXX} {saveContext XXX}`

Purpose: This saves all messages in a context into a SMAT database.

This Tcl proc opens a SMAT database at start mode, saves messages at run mode, and closes the SMAT database at shutdown mode.

- If `smatdbname` value is not provided, then a warning is reported and there is no save.
- If `saveContext` is not given, then the TPS context/state is used.

Resend and trace

You can resend a message of a user-defined SMAT database into an existing queue using the `hcicmd` command.

Trace information is stored into the tracing database as a new record in the `smatdbinsert` command.

The type of this new trace record is TPS, which is an already-existing trace type. The SMAT database name is saved into the SMAT column.

The TPS type and SMAT column are not null and indicate that this record is saved by the SMAT API.

The sequence ID determines the order of SMAT records if a messages is saved into multiple SMAT database files that are opened in one TPS.

Usage examples

Tcl example:

```
switch -exact -- $mode {
global smatdbTest
  start {
    if {[info exists smatdbTest]} {
      set smatdbTest [smatdbopen test.smatdb user_defined]
    }
  }
  run {
    keylget args MSGID mh
    set retCode [smatdbinsert $ smatdbTest $mh]
    if { retCode != 0 } {
      set errorContext "Failed to insert msg. Error code is $retCode"
      lappend dispList "ERROR {{$mh $ errorContext}}"
    }
  }
} else {
  set cycleRet [smatdbcycle $ smatdbTest ]
  if { $ cycleRet == -1 } {
    puts " cycle failed"
  } else {
    puts " cycle success"
  }
}
set searchRet [smatdbsearch $ smatdbTest "Time > 1456914420000 offset 0 limit 3"] lappend dispList "CONTINUE $mh"
}
time {
}
```

```

        shutdown {
            smatdbclose $smatdbTest
        }
    }
}

```

Java UPoC example:

```

    public DispositionList process (CloverEnv cloverEnv, String context, String mode, Message
msg) throws CloverleafException {
        DispositionList dl = new DispositionList();
        if (mode.equalsIgnoreCase("start")) {
            SMATDB smatdbTest = new SMATDB(CloverEnv, "test.smatdb");
            smatdbTest.open();
        } else if (mode.equalsIgnoreCase("run")) {
            if (smatdbTest.insert(msg) != 0) {
                String errorContext = "failed to insert a message"
                dl.add( DispositionList.ERROR, msg, errorContext);
            } else {
                smatdbTest.cycle();
                String searchRet = smatdbTest.search(" Time > 1456914420000 offset 0 limit
3");
                dl.add( DispositionList.CONTINUE, msg);
            }
        } else if (mode.equalsIgnoreCase("shutdown")) {
            smatdbTest.close();
        }
        return dl;
    }
}

```

Default saveContext for each TPS

This tables shows the available saveContext for each TPS:

TPS name	saveContext
1 TPS inbound data	sms_ib_data
2 TPS inbound reply	sms_ib_reply
3 TPS outbound data	sms_ib_reply
4 TPS outbound reply	sms_ob_reply
5 Xlate raw procs	xlt_raw
6 Xlate pre procs	xlt_pre
7 Xlate post procs	xlt_post
8 Xlate gen procs	xlt_gen
9 Prewrite procs	prewrite
10 Send OK procs for data	send_data_ok
11 Send Fail procs for data	send_data_fail
12 Send OK procs for reply	send_reply_ok
13Send Fail procs for reply	send_reply_fail

TPS name	saveContext
14 Reply generation procs	reply_gen
15 Upoc read tps	pdupoc_read
16 Upoc write tps	pdupoc_write
17 Protocol start-up procedures	proto_startup
18 fileset/ftp inbound deletion	fileset_ibdel
19 fileset/ftp inbound directory parse	fileset_ibdirparse
20 http client query tps	httpc_query

Table smat_msgs definition

This table shows the table `smat_msgs` data types:

Message definition	Data type
1 MessageContent	BLOB
2 DataLen	INTEGER
3 DestConn	VARCHAR
4 OrigDestConn	VARCHAR
5 DataFmt	VARCHAR
6 SepChars	VARCHAR
7 DriverCtl	VARCHAR
8 UserData	INTEGER
9 MidDomain	INTEGER
10 MidHub	INTEGER
11 MidNum	INTEGER
12 SrcMidDomain	INTEGER
13 SrcMidHub	INTEGER
14 SrcMidNum	INTEGER
15 Type	VARCHAR
16 SourceConn	VARCHAR
17 OrigSourceConn	VARCHAR

Message definition	Data type
18 Time	INTEGER
19 Priority	INTEGER
20 SaveContext	VARCHAR
21 Flags	INTEGER
22 State	INTEGER
23 GroupMidDomain	INTEGER
24 GroupMidHub	INTEGER
25 GroupMidNum	INTEGER
26 XltThread	VARCHAR
27 Retries	INTEGER
28 kipXlt	INTEGER
29 UseRecoverDB	INTEGER
30 GroupID	INTEGER
31 TimeIn	INTEGER
32 TimeXlt	INTEGER
33 TimeOut	INTEGER
34 TimeQCur	INTEGER
35 TimeQTot	INTEGER
36 TimeRec	INTEGER
37 TimeStor	INTEGER
38 TimeArc	INTEGER
39 DBTable	VARCHAR
40 Flags_Resent	INTEGER
41 Flags_Trace	INTEGER
42 ErrorString	VARCHAR

SMAT database columns

There are 42 columns for SMAT DB.

See [smat_msgs table columns](#).

Each bit of `columnsBitMask` represents whether to query a column:

"1" means querying the column. "0" means no query.

From lowest to highest bit, it represent these columns in sequence:

- `messageContent;dataLen;DestConn;OrigDestConn;DataFmt;SepChars;`
- `DriverCtl;UserData;MidDomain;MidHub;MidNum;SrcMidDomain;`
- `SrcMidHub;SrcMidNum;Type;SourceConn;OrigSourceConn;Time;Priority;`
- `SaveContext;Flags;State;GroupMidDomain;GroupMidHub;GroupMidNum;XltThread;`
- `Retries;SkipXlt;UseRecoverDB;GroupID;TimeIn;TimeXlt;TimeOut;`
- `TimeQCur;TimeQTot;TimeRec;TimeStor; TimeArc;DBTable;Flags_Resent;`
- `Flags_Trace;ErrorString;`

Example:

<code>messageContent</code>	<code>0x0000000000000001</code>
<code>dataLen</code>	<code>0x0000000000000002</code>
<code>DestConn</code>	<code>0x0000000000000004</code>
<code>DataFmt</code>	<code>0x0000000000000008</code>
<code>SepChars</code>	<code>0x0000000000000010</code>
<code>DriverCtl</code>	<code>0x0000000000000020</code>
<code>UserData</code>	<code>0x0000000000000040</code>
<code>....</code>	
<code>ErrorString</code>	<code>0x0000020000000000</code>

To query `messageContent`, `dataLen`, and `DestConn`, the column bit mask is `0x0000000000000007`.

Using the SMAT API

The SMAT API can be used to save a message in other TPS contexts. As in the traditional SMAT database, you can write messages to a SMAT database at other locations in the engine; not only on the inbound and outbound protocol threads.

In the Tcl SMAT API, the user-defined SMAT database has these features (compared with the traditional IB/OB SMAT database):

- Full operations
- Multiple writers
- Resend mapping
- Trace storage

These behaviors are shared between the two types of SMAT database:

- Table schema is the same.
- SMAT history and cycle logic share the same configuration with the traditional IB/OB SMAT database.
- Encryption key is the same as the SMAT database, that is, the site name.

Additional points include:

- Tcl and Java TPS are supported for the SMAT API.
- External Tcl scripts, for example, Tcl in an `hcitcl` shell, are also supported.

- The Java TPS is through Tcl, not natively.
- `xlt` and `jxlt` are not supported.

Tcl syntax and commands

The Tcl syntax is:

```
command arg1 ?arg2? -arg3... | arg4
```

In this command syntax:

- `?` (question mark) indicates the argument is optional.
- `-` dash indicates options.
- `. . .` (ellipse) indicates "and so on".
- `|` (pipe symbol) indicates "or".
- `<variable>` indicates a variable is required.

To create/open a SMAT database, specify:

```
smatdbopen smatdbName ?-e msgEncoding?
```

By default, the SMAT files are created in the process directory.

To store a message to a SMAT database, specify:

```
smatdbinsert smatdbHandle msgID ?saveContext?
```

If the SMAT (Save file) file is not open, then you must open the SMAT (Save file), before inserting a message.

You cannot insert a message from the TCL SMAT API to the traditional SMAT files.

This table lists the save contexts:

TPS name	saveContext
TPS inbound data	sms_ib_data
TPS inbound reply	sms_ib_reply
TPS outbound data	sms_ob_data
TPS outbound reply	sms_ob_reply
Xlate raw procs	xlt_raw
Xlate pre procs	xlt_pre
Xlate post procs	xlt_post
Xlate gen procs	xlt_gen
Prewrite procs	prewrite
Send OK procs for data	send_data_ok
Send Fail procs for data	send_data_fail

TPS name	saveContext
Send OK procs for reply	send_reply_ok
Send Fail procs for reply	send_reply_fail
Reply generation procs	reply_gen
UPOC read tps	pdupoc_read
UPOC write tps	pdupoc_write
Protocol start-up procedures	proto_startup
fileset/ftp inbound deletion	fileset_ibdel
fileset/ftp inbound directory parse	fileset_ibdirparse
http client query tips	httpc_query

Cycling and closing a SMAT database

To cycle, use:

```
smatdbcycle smatdbHandle
```

To close, use:

```
smatdbclose SMATDBHandle
```

Notes:

- If the SMAT (Save file) file is not open, then you must open the SMAT (Save file), before cycling.
- The SMAT handle points to the new SMAT file.
- There can only be one old SMAT (Save file) at a time.
- SMAT history does not effect the Tcl SMAT API. You must ensure the old SMAT files are archived.

Using shell commands

Start the respective interactive TclX command interpreter. To do this, specify `hcitcl` at the command prompt.

The interpreter then displays the `hcitcl>` prompt.

These shell commands are used before running `hcitcl`:

- To set the root, specify:
`setroot [rootdir [site]]`
- To set the site, specify:
`setsite site`
- To show the root, specify:
`showroot`

Searching for and deleting message in the SMAT database

To search for a message, use:

```
smatdbsearch smatdbHandle ?searchCondition? ?columnBitMask?
```

Note: If the SMAT (Save file) file is not open, then you must open the SMAT (Save file), before searching.

To delete a message, use:

```
smatdbdelete smatdbHandle ?deleteCondition?
```

Configuration tools

The runtime tools control and monitor connections, processes, and system information. The tools are:

- **Alert Configurator**
Use the Alert Configurator to set alarms or monitor points for user-notification when defined conditions happen.
- **BOX Manager**
A Buildable Object eXchange (BOX) is file package consisting of a collection of related system configuration files. Through the IDE, a BOX is transferred from one system root to another system root. Every BOX has a manifest file to record its content, the time of creation, the version, and descriptive notes.
BOX also provides the ability to share configuration resources between system sites on different hosts.
- **Database Schema Configurator**
Use this tool to configure database connections in the Network Configurator and map database schemas in the Translation Configurator. With these supports, you can significantly reduce Tcl script writing and improve usability on database-related configurations.
- **Engine Output Configurator**
This tool controls the type and amount of output information that is generated by the engine as it runs threads and processes. This information is reported by the engine in a log file.
- **FRL Configurator**
Fixed-length Record Layout (FRL) Configurator defines how data is formatted within fixed-length record layouts.
An FRL definition represents a single record where all the fields are a fixed width. In the system, all FRL fields have at least one subfield. Subfields can be of variable length as long as they fit within a fixed length field.
- **Global Variables Configurator**
The Global Variables Configurator is similar to the Lookup Table Configurator. Global variable values can be masked and encrypted similar to lookup table entries.
- **HL7 Configurator**
Health Level 7 (HL7) is a standard for electronic data interchange in healthcare environments, with an emphasis on hospitals. HL7 refers to the highest level of the Open System Interconnection (OSI) model from the International Standards Organization (ISO). In the OSI conceptual model, communications software and hardware functions are separated into seven layers, or levels. The HL7 standard is primarily focused on the issues that happen within the seventh layer, or application level.
- **HPRIM Configurator**

HPRIM makes it possible to exchange data among health care professionals. For example, medical labs, clinical services and hospitals facilities, blood center, radiology facilities. HPRIM plays the role of HL7 in France.

- **HRL Configurator**
Hierarchical Record Layout (HRL) supports these types of FRL or VRL records in a single message:
 - Repeating records
 - Groups of records
 - Repeating groups of records
 - Optional groups of records
 - Nested groups of records
- **Import Script**
Use an external editor to create new or open existing Tcl files. Then, import the modified or generated Tcl script files back to the IDE using the Import Script tool.
- **JSON Configurator**
JavaScript Object Notation (JSON) is an open standard format that uses human-readable text to transmit data objects consisting of attribute-value pairs. JSON is a language-independent data format. It is the primary data format used for asynchronous browser/server communication.
- **LDL Configurator**
DICOM contains length-delimited fields. The Length Delimited Layout (LDL) tool specifies a DICOM record layout.
Standard DICOM transactions are available in the root and can be edited with the LDL tool.
- **Lookup Table Configurator**
Use the Lookup Table Configurator to configure the lookup tables used to convert message data. Lookup tables perform code translations within one or more translation configurations.
A Lookup Table is called up within a translation configuration and used in the Table action for a specific field.
- **NCPDP Formulary and Benefit Configurator**
To better support e-Prescribe, message format support of NCPDP SCRIPT and NCPDP Formulary and Benefit is included.
The NCPDP Formulary and Benefit standard defines a file format by which a payer publishes formulary information to a consumer. Consumers are entities such as providers or clearinghouses. The NCPDP Formulary and Benefit standard is batch-oriented, being essentially a data push from the publisher to the consumer. There is no interactive use case. A small batch message flows back from the consumer to the publisher. This indicates the correctness of the file and any errors that have happened.
- **NCPDP SCRIPT Configurator**
The NCPDP SCRIPT standard defines messages for placing and managing electronic prescriptions. The partners that are involved in SCRIPT transactions are a prescriber, such as a doctor, and a pharmacy which fills the prescriptions.
NCPDP SCRIPT Configurator defines data elements, composite data structures, data segments, and messages for the NCPDP SCRIPT standard.
- **NCPDP Telecom Configurator**

National Council for Prescription Drug Programs (NCPDP) defines a set of standard message formats used in the prescription drug industry. These formats address messages from pharmacy to payer, prescriber to pharmacy, and others. The NCPDP Telecom formats that have been designated as HIPAA standard transactions are supported.

- Network Configurator

Use the Network Configurator to define and modify network connections, and the actions to take when handling communication transactions flowing across those connections.

Configure all connections from the viewpoint of the system site, which is assumed to be the starting point for any connection.

- Script Editor

The Script Editor is a built-in editor where you can create, open, and save scripts through the host server.

Use Script Editor to customize many aspects of the engine and how messages are handled in the system. This is accomplished by creating files for storing Tcl procedures and editing existing files or Tcl procedures.

- Site Documentation

The IDE is designed to configure a system site, and you review system sites through the IDE. For more flexibility and convenience, you can also remotely review system sites through the Site Documentation feature.

Site Documentation shows the available configurations in a specific site using HTML pages by collecting the important site information and configurations, displaying the details, and providing the links to the available configurations.

- Translation Configurator

The Translation Configurator is used to define how incoming records are translated into outgoing records.

The translation thread:

- Takes the message.
- Parses the record.
- Translates it into a new format and, often, a new protocol.
- Routes it to the destination.

- UN/EDIFACT Configurator

United Nations Electronic Data Interchange For Administration, Commerce, and Transport (UN/EDIFACT) was created from a requirement for a single international Electronic Data Interchange (EDI) standard. This standard is flexible enough for governments and private industries.

UN/EDIFACT Configurator defines data elements, composite data structures, data segments, and messages for the UN/EDIFACT standard.

- VRL Configurator

Variable-length Record Layout (VRL) Configurator defines how variable-length data is formatted within flat-record layouts.

VRL is similar to FRL in structure and purpose, but the underlying approach to accessing the data is different.

- WSDL2XSD

The XSD WSDL tool is integrated into the IDE as a wizard. The wizard guides you in generating and compiling the XSD file, and generating the WSDL file from the XSD file.

- X12 Configurator

An X12 message is defined as a single transaction set. It is a batch subsystem that parses and encodes, dealing with interchanges and groups.

In system terms, a batch refers to a group of one or more messages. These messages are surrounded by header and trailer segments with no special characters required to delimit the individual messages inside the batch.

- XML Package Manager

Extensible Markup Language (XML) is a general purpose markup language. This is widely used across the world to share structured data across various information systems. Support is provided for XML as a built-in message format.

Alert Configurator

Use the Alert Configurator to set alarms or monitor points for user notification when defined conditions happen.

The **Site Preferences > Alert Configuration** tab is used to configure alert-related preferences.

The default alert location is at `default.alrt`. Each site contains a file. For example:

```
C:\cloverleaf\cisversion\integrator\sitename\Alerts\default.alrt
```

Alert definitions in the alert file include all parameters configured in the GUI.

The daemon monitors the alert configuration files for modifications and reloads the file when any modification is detected.

An alert that is armed, or depends on some previous value, does not have its state changed by a reload if its configuration is unchanged.

Configuration validation

`hcialertcheck` checks for errors, as continuing operation after an error has been reported can lead to unexpected alert firings or missed alerts.

When using the command line, this script must be run within a site after `setroot` and `setsite` have been run.

```
hcialertcheck [-f alertFile]
```

This command validates the alert configuration file *alertFile*.

If *alertFile* is not found in `$HCISITEDIR/Alerts`, then the script throws an error.

If *alertFile* is not specified, then the command checks `default.alrt`.

Errors prevent the alert from loading. The alert loads with warnings, but the alert as defined might not run as expected.

Alert types

Select specific alerts to set alarms or monitor points for notification when defined conditions happen.

and alert

This alert monitors any logical operation combining other named alerts.

Clicking the **Source** button opens the **Select And Alert Source** dialog box, where you configure the alert. At first, the dialog box is blank. In this case, click **Add** to open the **Select Alerts** dialog box. Otherwise, the table lists the alert names the "and" alert is combining and the and or nand logical operation for each alert.

This dialog box lists the names of all available named alerts within the current site and master site. The list of alert names is sorted in ascending order and is case-insensitive. Master site names are shown in italic font.

Select the check box in the "Not" column to specify the alert is in nand logical operation; or, clear the check box to specify the alert is in and logic operation.

All "and" alerts are run last in a polling of the alerts. The value of any source has been updated before it runs. The order of running several "and" alerts is sorted by definition so that the alert's sources are run before the referencing alert.

Because the alert specification is stored in the alert file, any alert file name conflict is handled before the alert name conflict. When a master site alert file has a name conflict as locals, the alerts that are defined in the master site alert file are disregarded.

The **Source** field is space-delimited.

If the alert name has spaces, then the alert name and corresponding logical operation are surrounded with curly brackets.

The **Comparing** operations for this alert are == or !=. The only values to be compared are true or false.

Runtime errors disable the "and" alert. For example, when a source is not found among the alerts, or a circular reference is found.

In this example "B" references "C" which references "B":

```
Alert Name = "B"
Alert Type = "and"
Source = "C"
Source Count = "all"
Comparing = "==" and "false"
Duration = "once"
Action = "notify"

Alert Name = "C"
Alert Type = "and"
Source = "B"
Source Count = "all"
Comparing = "==" and "true"
Duration = "N minutes" and "30"
Action = "notify"
```

This results in an unexpected alert behavior and is logged as a warning.

disk % full

This states the percent of disk space that is used in the specified file systems.

See [Thread status and disk % full](#).

Defining an alert for disk % full

This example shows using the Alert Configurator to run a script if disk space is more than 90% full for at least 30 seconds.

Note: If your process configuration uses **Disk-based Message Queuing**, then use an alert for 90% full disk space.

- 1 On the Alert Configurator, click the **New Append** tool.
- 2 In the Trigger pane, select "disk % full" from the **Alert Type** drop-down menu.
- 3 Clear **Delta** to check the actual percentage of full disk space.
- 4 In **Source**, specify the path of the root directory to monitor after the drive letter
For example, path names should be in the form of: C:\cloverleaf\cis6.2\integrator
Or, you can click the button to open the **Select File System** dialog box, where you select the drive. Click **Apply** and the selected drive displays in the **Source** field, where you can finish specifying the path.
- 5 Click **Apply**.
- 6 Set **Source Count** to **All** to know when the file system approaches capacity. This is the default.
- 7 Set **Comparing** to **>** and specify **90** in the adjacent text box for the alert condition to trigger when the disk space is more than 90% full.
- 8 Set **Duration** to **N Seconds** and specify **30** in the adjacent text box to have the alert trigger when disk space remains more than 90% full for 30 seconds.
- 9 Click **Apply**.
The next alert triggers two actions:
 - First, a shell script (UserCleanup, in this example) runs, notifying users to clean up the directories.
 - Then, a message is sent to root at the specified trigger.
- 10 In the Actions pane, select **exec** from the **Alert Action** drop-down menu. The pane reconfigures for command input.
- 11 Specify the name of the script (UserCleanup) in the **Command** text box.
- 12 Click **Update**.
- 13 Click **Apply**.
- 14 On the Alerts list of the Alert Configurator, click the alert configured, if necessary.
- 15 Click the **Append** tool in the Actions pane.
- 16 Select the **exec** alert action.
- 17 Specify in **Command**:
`mailx -s '%A' root < \dev\null (UNIX)`
This sends a message to root with the "disk % full" subject line and an empty message body.
- 18 Click **Update**.

19 Click **Apply**. The new alert configuration displays on the Alerts List.

Thread status and disk % full

This example shows using the Alert Configurator to define alerts that notify the operator if any thread goes down. It then runs a script if disk space is more than 90% full for at least 30 seconds.

Note: If your process configuration uses Disk-based Message Queuing, then use an alert for 90% full disk space.

- 1** Open the Alert Configurator and click **New Append**.
Use the default **thread status** alert.
- 2** In the Trigger pane, select the thread names to monitor in Source. Click the button to select the thread names from a list of all threads that are configured for the system.
- 3** Click **Apply**.
- 4** Set **Source Count** to **At Least** and specify **1** in the text box to the right. Do this to see if any threads enter the down state.
- 5** In **Comparing**, select **==** from the menu. This is the default.
- 6** Then select **down** in the Comparing .
- 7** Set **Duration** to **Once**. This is the default. The example does not specify that the alert condition has to exist for a certain period of time. The triggers as soon as a thread enters the down state.
- 8** Click **Apply**.
- 9** In the Actions pane, select **notify**. This is the default. This sets the Alert Action to notify the operator when the alert triggers.
- 10** Click **Update** and then click **Apply**. The updated alert displays on the Alerts List.
- 11** On the Alert Configurator, click the **New Append** tool.
- 12** In the Trigger pane, select **disk % full** from the **Alert Type** menu.
- 13** Clear **Delta**, if required, to check the actual percentage of full disk space.
- 14** In **Source**, specify the path of the root directory to monitor after the drive letter.
For example, path names should be in the form of: C:\cloverleaf\cis6.2\integrator
Or, you can click the button to open the **Select File System** dialog box, where you select the drive. Click **Apply** and the selected drive displays in the **Source** field, where you can finish specifying the path.
- 15** Click **Apply**.
- 16** Set **Source Count** to **All** to indicate when the file system approaches capacity.
- 17** Set **Comparing** to **>** and specify **90** in the adjacent text box. This triggers the alert when the disk space is more than 90% full.
- 18** Set **Duration** to **N Seconds** and specify **30** in the adjacent text box. This triggers the alert when disk space remains more than 90% full for 30 seconds.
- 19** Click **Apply**.
The next alert triggers two actions. First, a shell script runs. In this example, it is `UserCleanup`. This notifies users to clean up the directories. Then, a message is sent to root at the specified trigger.
- 20** In the Actions pane, select **exec** from the **Alert Action** menu. The pane reconfigures for command input.
- 21** Specify the name of the script (`UserCleanup`) in the **Command** text box.

- 22 Click **Update**.
- 23 Click **Apply**.
- 24 On the Alerts list of the Alert Configurator, click the alert configured, if necessary.
- 25 Click the **Append** tool in the Actions pane.
- 26 Select the exec alert action.
- 27 In **Command**, specify `mailx -s '%A' root < \dev\null (UNIX)`.
This sends a message to root with the "disk % full" subject line and an empty message body.
- 28 Click **Update**.
- 29 Click **Apply**. The new alert configuration displays on the Alerts List.
- 30 Click **Save** on the Alert Configurator toolbar after configuring all alerts. Or, select **File > Save As** to open a file selection dialog box.
- 31 Specify a file name, or select from the listed files and click **Save**.

disk free space

This lists the amount of free disk space, in kilobytes.

disk IO per second

This lists the number of disk I/O operations per second on the specified hosts or disk name.

engine status

This specifies the engine process status. An engine is running or dead.

error count

This lists the number of transaction errors for the specified threads.

See [error database alert](#).

error database alert

This monitors the number of messages in the error database.

The "error count" alert looks at the error count statistic for the thread as stored in shared memory. This is the count that shows up in the GUI under "failed" when you look at a thread's status.

The "error database" alert looks at the number of messages in the error database that arrived from the given thread.

How these numbers differ:

- You can change one number without changing the other. For example, you can reset monitoring statistics, and therefore the "error count" alert, without affecting the error database when using `hcimsiutil`. You can also change the number of messages in the error database, therefore changing the "error database" alert. This does not affect monitoring statistics by resending messages or reinitializing the database.
- Messages in the error database do not necessarily come from a protocol thread. If you attempt to create an "error database" alert, then you can select translation threads as the source for the alert. For example, you can create a raw route and use the `xlteError` procedure to generate errors. If you set your alert to trigger on the protocol threads, then the alert never fires. If you set the source to be the translation thread, then the alert fires.

Therefore, select which alert to use depending on what to monitor:

- Use the error database alert to monitor error database depth.
- Use the error count alert to measure how many messages fail going through a protocol thread.

The source value is a set of thread names. The dialog box for configuring the source is the same as other thread alerts, such as thread status and protocol status.

Click **Source** to open the **Select Alert Source** dialog box.

This dialog box provides a choice for applying the alert to all threads or specific threads.

Selecting the **All Threads** option implies that all error messages in the error database are selected. In this case, all existing thread names are shown in the source field.

file change

This tracks when a file changes.

file size alert

This alert type is used to monitor the size of a file in kilobytes.

When file size is the **Alert Type**, **Source** is populated with the file location.

Click the **Source** button to open a file browser, where you select the file to monitor.

```
{SOURCE {file location}}
```

file location is the absolute path for the file to be monitored.

Multiple source values are not supported for this alert type. If more than one source value is specified, then the alert is loaded with a warning and only the first source is valid.

The value to be compared must be the size of the file in kilobytes. If the file does not exist, then the value that is used in the **Comparing** operation is -1.

The comparing operation is a numerical comparison, such as < N, <= N, >= N, > N, == N, != N, between N1 and N2, or diff.

forward count

This lists the number of messages forwarded for the specified threads.

idle CPU %

This states the percent of CPU time spent in idle cycles on the specified hosts.

inbound latency

This specifies the inbound message latency for the specified threads.

This is the elapsed time in milliseconds from when an inbound message is read by the engine to when translation is complete. This alert can be configured with delta mode to monitor whether an inbound message is processed for a long time.

inbound queue depth

This lists the number of messages in the inbound queues of the specified threads.

last receive

This states the time that the last inbound message was received from the specified threads.

last send

This states the time that the last outbound message was sent to the specified threads.

last update alert

This alert is the same as other existing time alerts. For example, "last receive" and "last send".

This alert checks the last update time of the thread data in shared memory. This can be used to monitor if a thread or engine is hanging, where the process is running but internal operations have halted. Such a condition may stop the normal thread updates of the shared memory data that happen about every five seconds.

The contents and options of this alert are the same as the other existing time alerts. In some ways, this alert overlaps the other time alerts but can be much quicker. The duration requires only 30 seconds to ensure that there is a problem.

In the case of a "zero" last update time (that is, the shared memory has never been updated), the alert is skipped.

Note: This alert fires if the process is dead. To avoid false or duplicate alerts you can combine its state with the process state using an "and" alert.

In CIS 6.0 and earlier versions, the current last receive alert would get the value of "MSI last receive" even though it was disabled. Because the value of the alert had been updated, it was triggered when it was enabled.

This was caused by the alert not resetting on a disable. When it was enabled, it would fire in zero time if the limit was exceeded.

This issue is fixed by having the last sample time of the alert be saved as the last time when disabled more than up to five seconds. This is the MonitorD sample time. When enabled, that time is used for the alert test until the new time in MSI. Other conditions are if the value in MSI was reset or the thread was restarted.

For correct operation of these alerts, this cannot be accomplished from the Alert Configurator GUI.

The **Activate/Deactivate** options on the Alert Configurator GUI use a different logic from the `hcicmd` enable/disable commands.

You must use `hcicmd` to enable/disable the "last update," "last receive," and "last send" alerts.

license status

This specifies the license status that is measured in the number of days remaining before expiration.

message archiving alert

This monitors how many messages are waiting to be archived for the table named in Source.

This alert defaults to sampling on the first pass after the daemon starts and every three minutes thereafter. To change the sampling interval, use the `dbrate` daemon command.

This is a single source alert. Additional sources create an error.

The **Source** has one choice: `TIMEOUT`.

In this mode, the **Comparing** field is the number of seconds because the oldest message in the message archive was written to the cache. That is, if there is a message that is waiting to be written to the external database for 20 minutes, the value would be 1200. All numerical comparisons are valid.

Note: Message archiving became internal starting with CIS 6.0 and is configured at Network Configurator. The MonitorD automatically loads and detects any changes to this configuration. You should not create your own message archiving alerts.

outbound latency

This specifies the outbound message latency for the specified threads. This is the elapsed time in milliseconds from when an outbound message is queued for transmission to when it is successfully delivered to the destination host. This alert can be configured with delta mode to monitor whether an outbound message is processed for a long time.

outbound queue depth

This lists the number of messages in the outbound queues of the specified threads.

physical memory free

This lists the free (available) real memory space on the specified hosts, in kilobytes.

Physical memory is for working processes. Linux puts idle processes onto the SWAP memory. This means you can have a virtual memory that is almost full, and still have available space in the physical memory.

Both **Physical memory free** and **Physical memory % free** are divided into three sections:

- Physical memory (hardware or hard-drive partition)
- Buffer
- Cache

This is used when processes load documents. You can free this when the system is low on memory. You can also have a virtual memory that is almost fully used. The available space is increased by flushing the cache.

physical memory % free

This is the percent of free (available) real memory space on the specified hosts.

Physical memory is for working processes. Linux puts idle processes onto the SWAP memory. This means you can have a virtual memory that is almost full, and still have available space in the physical memory.

Both **Physical memory free** and **Physical memory % free** are divided into three sections:

- Physical memory (hardware or hard-drive partition)
- Buffer
- Cache

Cache is used when processes load documents, and can be freed if additional memory is required. You also can have a virtual memory that is almost fully used. The available space can be increased by flushing the cache.

post-xlate queue depth alert

This monitors the sum of the queue depths of the post-xlate queues that are available through shared memory. Except for being the sum of the depths, this alert type is a normal queue depth type alert. This permits all options that can be used for other queue alerts.

The current depths for each destination thread are shown when you run `hcimsiutil`.

pre-xlate queue depth alert

This monitors the depth of the pre-xlate queue that is available through shared memory. This alert type is a normal queue depth type alert that permits all options that can be used for other queue alerts.

The current depth is shown when you run `hcimsiutil`.

process status alert

This alert monitors the MonitorD, Lock Manager, and all processes. Every site has the entire set of PID sources except for current site. This does not have the MonitorD because the PID alert runs in the MonitorD.

Clicking the **Source** field's button opens the **Select PID Alert Source** dialog box.

Select the nodes to indicate which ones are to be monitored. When selecting, if the node has any sub-nodes, the sub-nodes are automatically selected. If any of the sub-nodes are cleared, then the parent node is automatically cleared.

The **Source** field is read-only and shows the configured PID source.

```
{SOURCE {{{[SITE sitename] }[processtype processname]} ...}}
```

- *sitename* is the name of the site in with the process/daemon is found. This key is optional. If not specified, then the default is the current site.
- *processtype* is DAEMON OR PROCESS.
- *processname* is:
 - If *processtype* is DAEMON, then *processname* is `hcilockmgr` or `hcimonitor`. This is only available for another site.
 - If *processtype* is PROCESS, then *processname* is the `hciengine` process name.

Multiple `SOURCE` values are supported for this alert type.

The only operations for this alert are `==` or `!=`. The only values to be compared are "dead" and "running."

Note: If you are using an old alert file that contains any unknown statuses, then when modified it is saved as "dead."

recovery database

An issue could result when an ancillary goes down and a thread backs up the database. Setting an alert on every outbound thread to monitor the queue depth is not feasible when there are many threads.

For example, the number of messages in the recovery database nears the threshold that was set in the alert. After some time, though, the number drops below the threshold.

Because the alerts only periodically check the database, there is the possibility that the large increase in the number of threads was not recorded.

This would not be an issue since the number of threads dropped, nor is it an issue that the temporary increase was missed.

To ensure the queue does not get too large, you can now set one alert to monitor the recovery database, and not the individual threads.

To do this, use the **recoverydb** alert.

In this alert type, the **Source** can only be configured as **All Threads**. This is the only option on the **Select Alert Source** dialog box.

For example:

```
{ALERT
  { NAME recovery_db }
  { VALUE recoverydb }
  { SOURCE ALL_THREADS }
  { WITH -1 }
  { COMP {>= 1} }
  { FOR once }
  { WINDOW */*/*/ }
  { ACTION {
    { notify {} }
  } }
}
```

To monitor fields, click **Advanced** in the Actions panel to select fields to monitor from the **Advanced Message** dialog box.

When the alert fires, the specified alert fields and their values are displayed in the Advanced Message column of the **Alerts** dialog box.

system CPU %

This states the percent of CPU time that is used by system processes on the specified hosts.

tcl alert

This alert type monitors any condition that can be evaluated by a Tcl procedure.

When tcl is the **Alert Type**, the **Source** field is populated with the tcl alert definition.

Clicking the **Source** button opens the **Select Tcl Alert Source** dialog box.

thread held

This specifies whether an outbound message queue is held.

thread status

This specifies the status of the selected threads. An "up" thread is active. A "down" thread is inactive.

See [Thread status and disk % full](#).

total latency

This specifies the total message latency for the specified threads.

This is the elapsed time in milliseconds from when an inbound message is read by the engine to when the outbound message is delivered to the destination host. Because of waiting time in queues and other processing overhead, total latency is slightly larger than the sum of the inbound and outbound latencies.

This alert can be configured with delta mode to monitor whether a message is processed for a long time.

transactions/sec

This lists the number of transactions that are processed per second for the specified threads. This alert value is calculated by measuring the "xlate count" value over time, using: $[(\text{current xlate count}) - (\text{previous xlate count})] / [(\text{current poll time}) - (\text{previous poll time})]$

For example, the polling rate is 1 second. If the value is 15 for the first polling interval and 18 for the next polling interval, then the value of the alert is 3.

If the comparison value is an integer, then the value is computed using integer arithmetic.

If the comparison value is a float, then the value is computed using floating arithmetic.

user CPU %

This states the percent of CPU time that is consumed by user processes on the specified hosts.

virtual memory % free

This states the percent of free (available) virtual memory space on the specified hosts.

virtual memory free

Lists the free (available) virtual memory space on the specified hosts, in kilobytes.

wait CPU %

This states the percent of CPU time that is used in wait states on the specified hosts.

Deactivating and activating alerts

The Alert Configurator provides a way to enable and disable alert configurations.

Deactivated alerts are shown in the Site Document as unavailable.

- If no alert configurations are selected, then these options are disabled.
- If the selected alert configurations are activated, then **Activate** is disabled and **Deactivate** is enabled.
- If the selected alert configurations are deactivated, then **Deactivate** is disabled and **Activate** is enabled.
- If the selected alert configurations contain both activated and deactivated entries, then both items are enabled.

When a deactivated alert configuration is selected, its contents are shown on the property pane, where you can edit as necessary.

Alert Properties Trigger pane

Click to create and edit alerts.

Option	Description
Apply Alert Properties	<p>NewThis commits the changes. If this is not clicked before you go to another alert, then a dialog box opens asking to apply the changes.</p> <p>This button is disabled when Automatically apply alert properties changes is selected in Client Preferences > Alert Config.</p>

Option	Description
Delta	<p>This specifies the mode for interpreting alert values measured by the system. This is the difference between the current and previously measured value. This option displays only with numeric values (alerts).</p> <p>Clear this to use the actual alert value currently measured.</p> <p>Select the check box to measure the variable increment of the selected alert type. This is the difference between the current value of the selected alert type and the value previously measured.</p> <p>Note: The trigger pane only shows available fields for the current alert. Fields that cannot be applied in the current alert are hidden.</p>
Source	This specifies the resource to monitor.
Source Count	This specifies how many of the selected entities must satisfy the condition for the alert to trigger.
Comparing	This specifies the alert condition to test. The available options are dependent upon the selected Alert Type .
Duration	This specifies how frequently the alert condition must be satisfied to trigger the alert. Select the duration from the menu, specify the number (N) in the adjacent text box.
Repeating	This sets the duration period and maximum repeating number.
Time Window	This specifies a window of time when alert conditions are monitored.

Comparison options

For the Comparing option, select the comparison option symbol from the menu:

Option	Description
< (less than)	Measured value is less than the specified value.
<= (less than/equal)	Measured value is less than or equal to the specified value.
== (equal/equal)	Measured value is equal to the specified value.

Option	Description
>= (greater than/equal)	Measured value is greater than or equal to the specified value.
> (greater than)	Measured value is greater than the specified value.
!= (exclamation/equal)	Measured value is not equal to the specified value.
diff	Measured value is different from the last measured value.
between	Measured value is between the two specified values.
in	Measured value matches any one of the specified values.
timer	When configuring a protocol status alert, this sends an alert when a thread is in timer status.

Alert Properties Actions pane

Use the Actions pane to specify alert actions.

In **Alert Action**, select the action to take when the alert is triggered.

Action	Description
callback	This connects to a TCP/IP server application and sends the alert number and message. Specify the host name and socket port number. The dialog box reconfigures for Host and Port entry.
email	This is used to configure an email alert. This sends a user-defined message that is based on the alert. Use <code>hcitcl</code> to run Tcl scripts.
exec	This runs the specified script or command in the background. The dialog box reconfigures for command entry. You can run Tcl scripts using <code>hcitcl</code> .
none	This lets an alert fire but not run an external function run.
notify	This shows a message stating the alert is triggered.

Action	Description
tcl	<p>This runs the specified Tcl code. Click Edit to edit existing files or procs.</p> <p>Pathnames should be in the form of <code>C:\cloverleaf\cisversion\integrator</code>. Pathnames that include a backslash (\) are automatically replaced with a forward slash (/).</p> <p>The Tcl Interpreter does not return back to hcimonitor until it is finished. This indicates that hcimonitor does not do anything until the interpreter is finished processing your code. If you think your code could block or take several seconds to run, then use the exec action.</p> <p>The alert text or file name that contains the alert text can be passed to the tcl or exec actions.</p> <p>%A is the message that the alert engine would send if configured to notify. Because the message contains spaces, it must be passed inside quotes so that the Tcl routine sees it as a single argument.</p> <p>For example:</p> <pre>package require Alerts; Alerts::sendAlert "%A"</pre> <p>%F is the file name in the /tmp directory used by Alerts. This contains the %A message.</p>
Update	<p>This applies the current alert action and shows them on the action list.</p>

exec action

The **exec** action runs the specified script or command in the background. The dialog box reconfigures for command entry. You can run Tcl scripts using `hcitcl`.

Beginning in CIS 19.1, all **exec** actions must be approved in a allowlist before running the action. Internal built-in commands provided by Cloverleaf are already in the allowlist. External commands/scripts must first be added into the allowlist using the **Server Administration** tool (`hciserveradmin`).

External commands or scripts must include the full path. This is configured in the **alert** action.

hciuser profile

Beginning in version 5.8, the hciuser profile is a standard user, not an administrator. If your machine already has an existing hciuser, then there is no effect because it skips creating the user during installation.

This affects all commands when setting up the **exec** alert action on Windows.

To run a system command, `cmd /c` must be added before the command. For example:

```
cmd /c hciengineerun -p testprocess
```

To run the other commands, you must add the full path to the command. If a space is required inside the path, then double quotes must also be added around the command. For example:

```
"c:\test logger\bin\log.exe"
```

Note: The command which requests the GDI resource to show its dialog box cannot be used in the **exec** action. This is because the Windows standard user (hciuser) has no Desktop/GDI access.

tcl action

When the **tcl** action is selected, click **Edit** to open the Tcl Script Editor, where you can create, open, and save scripts.

The template for running the Tcl script is applied to the command field after clicking **Apply**.

Alert table

You can change the column order by dragging and dropping the column headers horizontally. Click a column header, drag it to the position, and release the button.

Note: The table column cannot be dragged outside the visible area of the alert table.

You can also sort columns by clicking on their headers. The default is that the Type column is sorted in ascending order.

Click the column header again for descending order. An "up" or "down" icon displays inside the column header. This indicates the order type and the table column that currently determines the sort order.

The display order and the displayed fields can be customized by right-clicking in the column header area.

When you first right-click a column header, the default fields to be shown are selected (Type, Name, Source, Trigger, and Actions).

Click **Customize** to list all available alert fields.

You can check the fields in the list to specify which fields are shown. You can also move up or down the fields to change the display order.

The default fields are **Type**, **Name**, **Source**, **Trigger**, and **Actions**.

Alert search

To search for a specific alert, use the text field on the toolbar.

The search criteria are "case insensitive" and "substring match" as in all other search functions in the IDE.

All properties in the alerts are compared when searched.

After a matching string is found, the whole alert row is selected and the first matched string in the alert property pane is highlighted.

Clicking **Search** again causes the next matched string to be highlighted.

Alert options

Alert options are configured on the trigger pane.

The **Alert Name**, **Group Names**, **Repeating**, and the **Time Window Not** options are applied for all alert types. These fields are always present.

Option	Description
Thread Only	<p>This option is only shown when an alert type is thread status, thread held, or protocol status. Select the check box for alerts with thread sources to arm and fire only when the process owning the threads is running.</p> <p>When selected, if the process in which the source thread resides is running, the alert works as code before the change. If the process is not running, then the alert is blocked from arming and resets if armed.</p> <p>This is useful when there are valid alerts to monitor the process that fire if required, and thread alerts would be redundant.</p>

Option	Description
Alert Name	<p>This is case-insensitive and can be empty. If it is not empty, then it must be unique in all named alerts that are defined in the current site. Every new alert has an auto-assigned name value (<i>Alert_ plus a numeric</i>).</p> <p>If the name is not unique, then an error message opens when the alert changes are committed.</p> <p>You can use this option to reference the alert by name in other alerts. There is an internal ID number assigned to each alert as it is loaded by <code>hcimonitor</code>. If there is no NAME key specified, then the name of the alert is assigned at runtime when the alert is loaded as:</p> <pre>anonymous alert #id</pre> <p><code>id</code> is the internal ID number, or ID.sub-ID as required, to create a unique name.</p> <p>You can also access the alert through the <code>hcicmd</code> interface.</p> <p>If there is a conflict between the default assigned name and an existing loaded alert, then an additional sub-ID is added. This happens until a unique name is found.</p>

Option	Description
Group Names	<p data-bbox="820 302 1409 527">This field is used for the enabling/disabling of alerts. To avoid false triggering, if you have alerts that are connected to a source that is down, then disable all alerts for that source. When the problem is resolved, you can enable those alerts. When the alerts for the source are in a group, this is a "one click" solution to enable or disable the alerts.</p> <p data-bbox="820 548 1409 840">Disable does not reset an alert, nor does it reset times that are used in alert tests. The time-out for the alert to fire is a "test time" of the alert. When it is enabled, that time is then used in the test. This continues until a new value is set by the thread or the thread is restarted. In general, do not clear the disable until you are sure that an event changes the time in msi or that the alert may fire. In most cases, this is after the thread has been bounced.</p> <p data-bbox="820 856 1370 915">Groups are enabled/disabled using the <code>hcicmd's disable</code> command.</p> <pre data-bbox="834 961 1127 982">disable group groupname</pre> <p data-bbox="820 1024 1390 1083">This disables alerts belonging to <code>group groupname</code>.</p> <p data-bbox="820 1100 1409 1192">Group names are configured through the Configure Group Names dialog box that opens when you click the button.</p> <p data-bbox="820 1209 1409 1302">A Group List is a collection of groups to which an alert belongs. An alert can belong to a maximum of 16 distinct groups.</p> <p data-bbox="820 1318 1409 1478">On this dialog box, you can add or delete groups. The Group field is editable, and optional. It lists all existing group names in the current site and master site. You can select one of the groups or specify a new group name.</p>

Option	Description
State From and To	<p>These fields tell the protocol status alert to filter which state changes arm the alert. The alert essentially becomes a "transition alert" that arms only on the state changes.</p> <p>If these options are not selected, then state changes are not filtered.</p> <p>These fields are only available when the Alert Type is protocol status and Comparing is diff.</p> <p>If the To field is bad, then state changes are filtered so that only those that match one of these arms the alert:</p> <ul style="list-style-type: none"> • * (any state): error • up: down • opening: down • up: opening <p>The State From field is also not available when bad is selected.</p>
Repeating	<p>When this is selected, the duration period and maximum repeating number are configured in the fields to the right. A repeating alert fires repeatedly until the alert is reset or the configured maximum number of firings is met.</p> <p>Use an integer greater than 0 to indicate the maximum number of times the action is to be repeated. Leaving Max empty indicates an unlimited number of firings.</p> <p>The maximum number refers to the number of repeat firings. That is, if you specify three repeats, then there are a total of four alert firings. These are the initial firing and the three repeats, if the condition remains true.</p> <p>The duration period value is N seconds or N minutes, but cannot be empty. Only integers greater than zero are permitted.</p> <p>If you select Repeating but do not specify any value for the duration period, then an error results.</p> <p>To specify an infinite number of repetitions, leave Max empty.</p> <p>When Repeating is selected, the duration period value is updated to reflect how long the condition has been true. This value is calculated using the difference between when the alert was triggered and the current firing time.</p>

Alert messages

An alert message contains a list of updated current values for alert types, when the alert message contains the current update value for alert types. Alert types include error count, inbound queue depth, file size, and others. Update values include idle CPU%, disk % full, disk free space, and others.

These values are the values that were available for the sources when the alert was fired. Some of the values might not be involved in causing the alert to fire.

For example, a queue depth alert that repeats every 60 seconds might have the first message be similar to:

```
Thread inbound queue depth of at least 2 of conn_1,conn_2,conn_3 has
  been more than 15 for 30 seconds. Currently, conn_1 is 20,conn_2 is 25,conn_3
  is 5.
```

The next updated message might be:

```
Thread inbound queue depth of at least 2 of conn_1,conn_2,conn_3 has
  been more than 15 for 90 seconds. Currently, conn_1 is 22,conn_2 is 29,conn_3
  is 12.
```

If an action needs to know the current repetition, then use the %R substitution characters.

Customized alert messages

You can append custom text to the alert message in the **Message** field.

You can also set up custom messages from within Cloverleaf so that they display on the GM server under the alert view. This includes % variables for alert values.

For example, when appending a message, there is {appended message} under ACTION:

```
{ALERT
  { NAME Alert_1 }
  { VALUE status }
  { SOURCE bouncefile }
  { WITH -1 }
  { COMP {== down} }
  { FOR once }
  { WINDOW */*/*/ }
  { ACTION {
    { notify {} {appended message} }
    { message {xxx} }
  } }
}
```

When the message is fully customizable, there is no {appended message} under ACTION :

```
{ALERT
  { NAME Alert_1 }
  { VALUE status }
  { SOURCE bouncefile }
  { WITH -1 }
  { COMP {== down} }
  { FOR once }
  { WINDOW */*/*/ }
  { ACTION {
    { notify {} }
    { message {message content} }
  } }
}
```

total latency

This specifies the total message latency for the specified threads.

This is the elapsed time in milliseconds from when an inbound message is read by the engine to when the outbound message is delivered to the destination host. Because of waiting time in queues and other processing overhead, total latency is slightly larger than the sum of the inbound and outbound latencies.

This alert can be configured with delta mode to monitor whether a message is processed for a long time.

Inserting variables in the tcl, exec, and email actions

A right-click menu is available in these actions:

- **Tcl code** field in the tcl action.
- **Command** field in the exec action.
- **Subject** and **Message** fields in the email action.

When you specify an exec, tcl, or email action, the text is scanned for the instance of these substitution characters:

- **%A** (alert message)
At runtime, this is replaced with the alert message.
- **%F** (temporary file name)
At runtime, this is replaced with the file name of a temporary file containing the alert message.
- **%G** (group membership)
At runtime, this is replaced with the alert's group membership list or "none" when no group exists.
- **%N** (alert name)
At runtime, this is replaced with the alert's name.
- **%R** (current repeat count)
At runtime, this is replaced with the current value of the repeat count, starting from zero, or "none", when no repeat.
- **%V** (current value)
At runtime, this is replaced with the current source value, a keyed list of sources and values, or "n/a" when there is no available current value.
- **%P** (all triggered processes in the current alert)
At runtime, this is replaced with all processes that triggered the alert at the "process status" or "engine status" alert type.
This is separated by a comma when there are multiple processes.
This is replaced with "n/a" when there is no existing process and other alert types, except "process status" or "engine status".
- **%T** (all triggered threads in the current alert)

At runtime, this is replaced with all threads that triggered the alert at the "thread status" or "thread held" alert type.

This is separated by a comma when there are multiple threads.

This is replaced with "n/a" when there is no existing thread or other alert types except "thread status" or "thread held".

- `%p` (process name of a thread-based (`%t`) alert that has triggered)

At runtime, when the alert type is "process status" or "engine status", it is replaced with the process that meets the triggering conditions.

When the alert status is "thread status" or "thread held", it is replaced with the process of the thread that triggered the alert.

This is separated with a comma when there are multiple processes.

This is replaced with "n/a" when there is no existing process or other alert types except "process status", "engine status", "thread status", or "thread held".

- `%t` (triggered thread)

At runtime, this is replaced with one thread that meets the triggering conditions at "thread status" or "thread held" alert types.

This is separated with a comma when there are multiple threads

This is replaced with "n/a" when there is no existing thread or any other alert type, except "thread status" or "thread held".

Note: These values are the values available for the sources when the alert is fired. Some of the values might not have been involved in causing the alert to fire.

Example

This example is about `%P`, `%p`, and is also valid for `%T` and `%t`.

For `%P` and `%p`, the alert type is "process status".

There are three processes P1, P2, and P3 configured in an alert.

If the **Source Count** is "any", then **Comparing** is "equal dead".

If P2 is "dead", then the alert is triggering. `%P` and `%p` are replaced with P2.

If P3 is "dead", then the alert is triggering again. `%P` is replaced with P2 and P3. `%p` is one of these two processes.

For `%p`, `%T` and `%t`, the alert type is "thread status". There are three processes, T1, T2, T3, configured in an alert and their processes are P1, P2, P3. If the **Source Count** is "any", then **Comparing** is "equal dead".

If T2 is "dead", then the alert is triggering. `%T` and `%t` are replaced with T2. `%p` is P2.

If T3 is "dead", then the alert is triggering again. `%T` is replaced with T2 and T3. `%t` is one of these two threads, if `%t` is replaced with T3, and `%p` is replaced with P3.

Substitution characters

For alert variables, right-click menus are available for actions. The selected variable is inserted at the cursor location.

The right-click menu is available only for these actions:

- **tcl** action: **Tcl code** field
- **exec** action: **Command** field
- **email** action: **Subject** and **Message** fields

These variables are available on the right-click menu:

Substitution character	Description	Action
%A	Alert	%A characters are replaced with the alert message. You can set up custom messages or replace the entire alert message, including % variables for alert values.
%F	Temporary file name	%F characters are replaced with the file name of a temporary file containing the alert message.
%G	Group membership	%G characters are replaced with the alert's group membership list. If no groups exist, then the two characters are replaced by the text "none."
%N	Alert name	%N characters are replaced with the alert's name.
%P	All processes in the current site	%P returns all triggered process names. The process status alert supports %P. "N/A" is returned if the processes cannot be found.
%p	Triggered process	Process name of a thread-based (%t) alert that has triggered.
%R	Current repeat count	%R characters are replaced with the current value of the repeat count, starting from zero. If no repeat was specified, then the two characters are replaced by the text "none."

Substitution character	Description	Action
%T	All threads in the current site	%T returns all triggered threads. %T is supported on all alerts, except process status. "N/A" is returned if the thread cannot be found.
%t	Triggered thread	%t is one triggered thread.

Substitution character	Description	Action
%V	Current value	<p>%V characters are replaced with the current source value or a keyed list of sources and values.</p> <p>If the current value of a source is not available, then the two characters are replaced by the text "n/a."</p> <ul style="list-style-type: none"> When the value is a delta, the delta is shown as the current value. <p>For these alert types, %V is replaced with the value keyed list:</p> <ul style="list-style-type: none"> transactions/sec forward count error count inbound queue depth outbound queue depth prexlate queue depth postxlate queue depth error database messages file size inbound latency outbound latency total latency last receive last send <p>For these single source alert types, %V is replaced by the numerical value:</p> <ul style="list-style-type: none"> tcl system CPU % wait CPU % user CPU % idle CPU % virtual memory % free virtual memory free disk I/O per second disk % full disk free space

Note: These values are the values available for the sources when the alert was fired. Some of the values might not have been involved in causing the alert to fire.

Customizing alert messages

Cloverleaf can deliver alert notifications in a variety of formats with customized alert messages.

You can customize the alert notification message by selecting from a list of predefined data elements. After making a selection, define a delimiter for the `csv` string alert message.

These data elements are available for building the alert message:

- Alert configuration
- Alert name (%N)
- Alert type
- Comparing
- Current repeat count (%R)
- Current time
This is the alert trigger time.

- Current value
- Duration
- Duration type
- Group name
- Host name
- Original alert message
- Root directory
- Site name
- Source
This is the thread name; not all sources.
- Source count
- Status
- Version

These elements are not available for building the alert message:

- Log-in user
- Password
- Port
- SMTP server

The **Alert Configurator** saves the chosen custom alert data element and delimiter to the alert configuration file, or a user-defined alert file in the same folder. The configuration file is located at `$HCISITEDIR/Alerts/default.alrt`.

MonitorD runtime loads its definition and populates the alert message accordingly when the alert is fired from the runtime.

MonitorD populates the values for the data elements in the **Custom Alert Message** dialog box.

A `CUSTOMMSG` key/value pair are generated and added to the alert definition for custom messages. For example:

```
{ALERT
  { NAME Alert_1 }
```

```

{ VALUE vmf }
{ SOURCE hostname }
{ MODE actual }
{ WITH -1 }
{ COMP {<= 30000} }
{ FOR once }
{ WINDOW {* * * * *} }
{ ACTION
  { notify {} {appended message for alert} }
}
{ CUSTOMMSG {
  {ITEM { AlertType AlertConf AlertName GroupName Source SourceCount Status Duration Dura
tionType
Comparing CurrentRepeatCount CurrentTime LastActiveTime SiteName Version RootDir Hostname Cur
rentValue
OriginalAlertMessage }}
  {DELIMITER keyvalue}
  } }
}

```

GUI configuration

The Actions pane of the Alert Configurator has an **Advanced** button. Clicking this opens the **Advanced Message** dialog box, where you configure custom messages.

When you create an alert message, you can select individual fields to add or remove from the custom alert message.

Click **Save** to save the custom alert message.

This table lists the areas and functions of the dialog box:

Function	Description
Alert Fields	Displays the predefined fields. No fields are selected by default.
Messages	Displays the selected items from Alert Fields . <ul style="list-style-type: none"> Double-click an item in Messages to move it to Alert Fields. Double-click an item in Alert Fields to move it to Messages.
←All	This is enabled when Messages has at least one item. Click this button to move all items in Messages to Alert Fields .
←	This is enabled when there is a selected item in Messages . Click this button to move the selected item in Messages to Alert Fields .
→	This is enabled when there is a selected item in Alert Fields . Click this button to move the selected item in Alert Fields to Messages .

Function	Description
All→	This is enabled if Alert Fields has at least one item. Click this button to move all items in Alert Fields to Messages .
Move Up	This is enabled when Messages has at least two items and the selected item is not the first one in Messages . Click this button for the selected item to exchange location with the previous one.
Move Down	This is enabled when Messages has at least two items and the selected item is not the last one in Messages . Click this button for the selected item to exchange location with latter one.
OK/Cancel	Click OK to save the customized alert message in Messages . Click Cancel to close the dialog box without saving.

The customized message is listed on Site Documentation when an alert is generated.

Alert configuration file

Alert configurations are stored in `$HCISITEDIR/Alerts/default.alrt`, or a user-defined alert file in the same folder.

The custom alert message is appended to the alert message content.

For example:

```
{ALERT
  {NAME Alert_1 }
  {VALUE vmf }
  {SOURCE hostname }
  {MODE actual }
  {WITH -1 }
  {COMP {<= 30000} }
  {FOR once }
  {WINDOW {* * * * *} }
  {ACTION
    { notify {} {appended message for alert} }
  }
  {CUSTOMMSG {
    {ITEM { AlertType AlertConf AlertName GroupName Source SourceCount Status
      Duration DurationType Comparing CurrentRepeatCount CurrentTime LastActiveTime
      SiteName Version RootDir Hostname CurrentValue OriginalAlertMessage }}
    {DELIMITER keyvalue}
  } }
}
```

This file contains the CUSTOMMSG key. This key has these sub-keys:

- ITEM

This lists all items that are defined for the custom alert message.

- **DELIMITER**

This has one fixed value and cannot be edited, and does not display in the **Advanced Message** dialog box. The fixed value is saved in the alert file.

This defines the literal alert message content. This is `keyvalue`.

Example: Configuring a custom alert message

User 1 is an Integration Engineer and is requested to add the process name to the up/down alert for the `adt_in` thread.

- 1 User 1 opens the **Alert Configurator** and creates a new thread status alert for the `adt_in` thread.
- 2 User 1 clicks the **Advanced** button in the Actions pane. This opens the **Custom Alert Message** dialog box.
- 3 User 1 sets the delimiter to `,`.
- 4 User 1 selects these fields for the alert message:
 - **Process**
 - **Source**
 - **Protocol Status**
 - **Current Time**
- 5 User 1 saves and tests the alert. This alert is received:

```
ADT, adt_in, Down, 03/19/yyyy 16:36:06
```

Alert Time Window dialog box

The **Alert Time Windows** dialog box supports both old and new alert pane specifications.

Click the **Time Window** button to open the dialog box.

Which specification is currently used is specified by clicking **By Range** or **By Event Time**.

By Event Time is for the old Windows specification.

The **Seconds** field is hidden, because this value is not configurable and must be `""`. Although this field is hidden, the **Seconds** value is still shown with the other time values in the **Time Windows** field. You can use `""` to correct the value. Otherwise, a warning is issued and the value ignored.

By Range, the default, is for the new Windows specification.

Click **New** to append a new time entry in the time table. The values for each time entry can be changed directly in the corresponding table cells.

The **Alert Time Windows** dialog box does not verify if the current configured time specifications are valid. To do this, you must use the **Validate** menu option.

By Range specification

For these options, using an asterisk (*) means "all", where any value is permitted.

Option	Description
Week Days	<p>Entries are listed by: * (all), weekday, weekday - weekdayWeek days are entered using the numbers 0 to 6, corresponding to Sunday to Saturday, or weekdaylist. This is a list of specific week days separated by a comma.</p> <p>To specify a week day range (weekday - weekday), the first week day must be before the second.</p>
Time Ranges	<p>For hours, use 0-23, which corresponds to a 24-hour day (12AM to 11PM).</p> <p>For minutes, use 0-59, which indicates the start of the minute to be included.</p> <p>Entries are listed by: * (all), timerange, or timerangelist. This is a list of specific times separated by a comma.</p> <p>To specify a time range (hour:minute - hour:minute), the first time must be before the second.</p>
Month Days	<p>Month days are entered using 1-31, corresponding to the days of the month.</p> <p>Entries are listed by: * (all), monthday, monthday - monthday, or monthdaylist. This is a list of specific month days separated by a comma.</p> <p>To specify a month day range (monthday - monthday), the first month day must be before the second.</p>
Months	<p>Months are entered using the numbers 1 to 12, corresponding to January to December.</p> <p>Entries are listed by: * (all), month, month - month, or monthlist. This is a list of specific months separated by a comma.</p> <p>To specify a month range (month - month), the first month must be before the second.</p>

Range Time Window dialog box

In the **Alert Time Window** dialog box, when you click within a field, a button displays.

Clicking the button opens the **Range Time Window** dialog box at the tab that corresponds to the selected type.

In this dialog box, you can customize the alert by specifying the alert conditions (week days, time ranges, month days, and months).

Alert example 1

This is an alert that is active from 7:30AM until noon, from 1PM until 4PM for Monday through Friday, and 10AM through 3PM on Saturday.

- 1 In the Alert Configurator, click the Time Window button to open the **Alert Time Window** dialog box.
- 2 In the **Alert Time Window** dialog box, click the button in a component field to open the **Range Time Window** dialog box.
- 3 Specify this information:
 - Week Days = 1-5
 - Time Ranges = 7:30 AM -12:00 PM, 1:00 PM -4:00 PM
 - Month Days = *
 - Months = *
- 4 Click **OK** to return to the **Alert Time Window** dialog box.
- 5 Add a new line by clicking **New**.
- 6 In the **Alert Time Window** dialog box, click the button in a component field to open the **Range Time Window** dialog box.
- 7 Specify this information:
 - Week Days = 6
 - Time Ranges = 10:00 AM -3:00 PM
 - Month Days = *
 - Months = *
- 8 Click **OK** to return to the **Alert Time Window** dialog box.
- 9 Click **OK** to finish.

Alert example 2

This example shows an alert that is active for all times except for the three listed U.S. holidays:

- 1 In the Alert Configurator, select the **Not** check box for Time Window parameter.
This indicates that the selected parameters are not included in the alert.
- 2 Click the Time Window button to open the **Alert Time Window** dialog box.
- 3 In the **Alert Time Window** dialog box, click the button in a component field to open the **Range Time Window** dialog box.
- 4 Specify this information:
 - Week Days = *
 - Time Ranges = *

- Month Days = 1
 - Months = January
- 5 Click **OK** to return to the **Alert Time Window** dialog box.
 - 6 Add a new line by clicking **New**.
 - 7 In the **Alert Time Window** dialog box, click the button in a component field to open the **Range Time Window** dialog box.
 - 8 Specify this information:
 - Week Days = *
 - Time Ranges = *
 - Month Days = 4
 - Months = July
 - 9 Click **OK** to return to the **Alert Time Window** dialog box.
 - 10 Add a new line by clicking **New**.
 - 11 In the **Alert Time Window** dialog box, click the button in a component field to open the **Range Time Window** dialog box.
 - 12 Specify this information:
 - Week Days = *
 - Time Ranges = *
 - Month Days = 25
 - Months = December
 - 13 Click **OK** to return to the **Alert Time Window** dialog box.
 - 14 Click **OK** to finish.

Alert file check

The **File > Validate** menu option on the Network Configurator opens the **Validation Results** dialog box. This validates the currently open alert file, showing the output in the dialog box.

Escalating an alert

Alert escalation is reporting a condition after it is addressed for some period of time.

Alert escalation can be accomplished by creating additional alerts that are exactly the same except for their duration and action values. These examples show several ways to define an alert escalation.

Alert escalation example 1

Alert name	Parameters
Main Alert Fired 15 Minutes	<ul style="list-style-type: none"> Alert Type: "thread status" Source: "inbound_1" Mode: "actual" Source Count: "all" Comparing: "==" and "down" Duration: "N Minutes" and "15" Action type: "tcl" Tcl code: <pre>{email "%N" "%A" Level1Support}</pre>
Main Alert Fired 60 Minutes	<ul style="list-style-type: none"> Alert Type: "thread status" Source: "inbound_1" Mode: "actual" Source Count: "all" Comparing: "==" and "down" Duration: "N Minutes" and "60" Action type: "tcl" Tcl code: <pre>{email "%N" "%A" TopLevelSupport}</pre>
Main Alert Fired 90 Minutes	<ul style="list-style-type: none"> Alert Type: "thread status" Source: "inbound_1" Mode: "actual" Source Count: "all" Comparing: "==" and "down" Duration: "N Minutes" and "90" Action type: "tcl" Tcl code: <pre>{callCIO "%N" "%A"}</pre>

Alert escalation example 2

This method creates "and" alerts that are based on the original alert.

Alert name	Parameters
Main Alert	<ul style="list-style-type: none"> Alert Type: "thread status" Source: "inbound_1" Mode: "actual" Source Count: "all" Comparing: "==" and "down" Duration: "once" Action type: "none"
Main Alert Fired 15 Minutes	<ul style="list-style-type: none"> Alert Type: "and" Source: "Main Alert" Mode: "actual" Source Count: "all" Comparing: "==" and "true" Duration: "N Minutes" and "15" Action type: "tcl" Tcl code: <pre>{email "%N" "%A" Level1Support}</pre>
Main Alert Fired 60 Minutes	<ul style="list-style-type: none"> Alert Type: "and" Source: "Main Alert" Mode: "actual" Source Count: "all" Comparing: "==" and "true" Duration: "N Minutes" and "60" Action type: "tcl" Tcl code: <pre>{email "%N" "%A" TopLevelSupport}</pre>
Main Alert Fired 90 Minutes	<ul style="list-style-type: none"> Alert Type: "and" Source: "Main Alert" Mode: "actual" Source Count: "all" Comparing: "==" and "true" Duration: "N Minutes" and "90" Action type: "tcl" Tcl code: <pre>{callCIO "%N" "%A"}</pre>

Alert escalation example 3

This method repeats the alert and creates multiple actions conditional on repetition.

Alert name	Parameters
Main Alert	<ul style="list-style-type: none"> Alert Name: "" Alert Type: "thread status" Source: "inbound_1" Mode: "actual" Source Coun: "all" Comparing: "==" and "down" Duration: "N Minutes" and "15" Repeating: "N Minutes" for "15" with a Max of "6" Action type: "tcl" Tcl code: <pre>{email "%N" "%A" Level1Support if {%R == 0}} {email "%N" "%A" TopLevelSupport if {%R == 3}} {callCIO "%N" "%A" if {%R == 5}}</pre>

Select Command dialog box

In the Alert Configurator, select the **exec** alert action and click **List** to open the **Select Command** dialog box. This shows all available commands.

The monitorD-related commands are shown in the MonitorD Commands section. Because these commands are a subset of the `hcicmd` command, they are placed under the `hcicmd` sub-section.

`hcialertcheck` is located in the Miscellaneous Commands section.

Starting the Monitor Daemon

After configuring the necessary alerts, start the monitor daemon with one of these methods:

- From the Launch Bar of the IDE, click the Site Daemons tool on the Runtime panel.
- From the Shell Window tool's command line, specify this command:

```
hcimonitor [-de eoc-pattern] [-dd eoc-pattern] [-cl cfg-list]
[-nl] [-D] [-S sitename]
```

If the Monitor Daemon was started with a saved alert configuration file, then the Monitor Daemon automatically rereads the file.

If the saved alert configuration file has another file name, then start the Monitor Daemon with the new file name.

For UNIX users, as the errdb alert type requires database access, during start-up the Lock Manager should be started before starting the Monitor Daemon. If an alert is triggered, then a message is displayed where the Monitor Daemon is running. For shutdown, the Monitor Daemon should be shut down first, followed by the Lock Manager.

- 1 Open the **Site Daemons**.
- 2 Select **Monitor Daemon**. This is the default and only available option on Windows.
On UNIX, select **Monitor Daemon** or **All Daemons**. (If `hcimonitor` is already running and there are changes to the alert configuration files, restart Site Daemons.)
- 3 Click **Alert Cfg**. This opens a selection dialog box.
- 4 Select the new alert configuration file and click **Apply**.
- 5 Click **Start Daemon** to start the Monitor Daemon.
- 6 Select **File > Exit** to exit the IDE, or right-click the **Site Daemons** tab and select **Close** to close the Site Daemons and remain in the IDE.

From the Site Daemons tool

- 1 Select **Monitor Daemon**.
This is the default and only available option on Windows. On UNIX, select **Monitor Daemon** or **All Daemons**.
- 2 Click **Alert Cfg**.
This opens a selection dialog box. Select the alert configuration file to process.
- 3 Click **Apply**.
The Site Daemons Status pane updates showing the new selection.
- 4 Click **Start Daemon** on the Site Daemons tool to start the Monitor Daemon.

From the command line

To start the monitor daemon from the command line, use `hcisitectl` or `hcimonitor`.

To start all daemons, use the `-s` argument.

To run a specific alert (in this example, the alert configuration file is named `foo.alert`), use this syntax:

```
hcisitectl -s a -A "a=-cl foo.alert"
```

.

hcisitectl

```
hcisitectl [-f] [-h host] [{ -K | -k daemon }] [-v]
[{ -S | -s daemon}] [-t timeout] [{ -R | -r daemon}]
-d delay interval [-u #users] [-A args] [-n]
```

- `-f` forces stopping the daemons.
- `-h host` is the name of the remote host.
- `-K` stops all daemon processes.
- `-k daemon` stops the specified daemons. For `daemon`, use a comma-separated list if specifying multiple daemons, where:
 - `l` is the Lock Manager.
 - `m` is the Monitor Daemon.
- `-v` checks for duplicate `hcimonitorD` when starting/stopping MonitorD, preventing duplicate MonitorDs in one site.
- `-s` starts all daemon processes.
- `-s daemon` starts specified daemons. For `daemon`, use a comma-separated list if specifying multiple daemons, where:
 - `l` is the Lock Manager.
 - `m` is the Monitor Daemon.
- `-t timeout` is the restarting delay, where `timeout` is the number of seconds to delay.
 - `hcisitectl -k -s -t timeout -m` applies the restarting delay to the MonitorD.
 - `hcisitectl -k -s -t timeout -l` applies the restarting delay to the Lock Manager.
 - `hcisitectl -K -S -t timeout` applies the restarting delay to all daemons.
- `-R` restarts all daemon processes and is used with `-d delay interval`.
- `-r daemon` restarts the specified daemon and is used with `-d delay interval`. For `daemon`, use a comma-separated list if specifying multiple daemons, where:
 - `l` is the Lock Manager.
 - `m` is the Monitor Daemon.
- `-d delay interval` is the time interval for engine restarting.
- `-u #users` is the maximum number of users for Lock Manager. `#users` defaults to 500 and must be greater than 200.
- `-A args` specifies the startup arguments for the daemons. For `args`, use a comma-separated list if specifying multiple arguments. Each entry has the form `<daemon> = args`, where:
 - `l args` is the Lock Manager, followed by arguments.
 - `m args` is the Monitor Daemon, followed by arguments.
 - `args` are the arguments to pass to that daemon.
- `-n` specifies to not run the engine in a service on Windows.

If no options are specified, then the status of the daemons is verified and reported.

Note: Do not stop the Lock Manager without first ensuring that there are no running engine processes in the site.

If `hcisitectl` is run with no options, then the status of the daemons is verified and reported.

hcimonitor

```
hcimonitor [-de eoc-pattern] [-dd eoc-pattern]
           [-cl cfg-list] [-nl] [-D] [-S sitename]
```

- `-de eoc-pattern` enables an engine output configuration pattern.
- `-dd eoc-pattern` disables an engine output configuration pattern.
- `-cl cfg-list` specifies a list of alert configuration files to process. Use a space to separate multiple file names in the list.
If no file name is specified, then `default.alert` is processed, if it exists.
- `-nl` enables non-daemon mode. This indicates no log file. In this case, the Monitor Daemon runs in the foreground.
This option is only used for debugging.
- `-D` enables debugging information.
- `-S sitename` is the site name.

Alert samples

These samples show how the alert types are used in customizing alerts. For example, protocol status, disk % full, inbound and outbound queue depth, and others.

Substitution characters are available when you create an alert. For example, when you specify an exec, tcl, or email action, the text is scanned for the instance of the `%sub` characters. If found, then the alert message text replaces the two characters. These substitutions permit the action to identify and get information about the alert that fired.

Right-click menus are available in the exec, tcl, and email actions that contain a list of alert variables. For example, using `%A` can be used to set up custom messages or replace an alert message. The `%A` is replaced with the alert message. You can set up custom messages or replace the entire alert message, including % variables for alert values.

Thread status and disk % full

This example shows using the Alert Configurator to define alerts that notify the operator if any thread goes down. It then runs a script if disk space is more than 90% full for at least 30 seconds.

Note: If your process configuration uses Disk-based Message Queuing, then use an alert for 90% full disk space.

- 1 Open the Alert Configurator and click **New Append**.
Use the default **thread status** alert.
- 2 In the Trigger pane, select the thread names to monitor in Source. Click the button to select the thread names from a list of all threads that are configured for the system.
- 3 Click **Apply**.

- 4 Set **Source Count** to **At Least** and specify **1** in the text box to the right. Do this to see if any threads enter the down state.
- 5 In **Comparing**, select **==** from the menu. This is the default.
- 6 Then select down in the Comparing .
- 7 Set **Duration** to **Once**. This is the default. The example does not specify that the alert condition has to exist for a certain period of time. The triggers as soon as a thread enters the down state.
- 8 Click **Apply**.
- 9 In the Actions pane, select notify. This is the default. This sets the Alert Action to notify the operator when the alert triggers.
- 10 Click **Update** and then click **Apply**. The updated alert displays on the Alerts List.
- 11 On the Alert Configurator, click the **New Append** tool.
- 12 In the Trigger pane, select **disk % full** from the **Alert Type** menu.
- 13 Clear **Delta**, if required, to check the actual percentage of full disk space.
- 14 In **Source**, specify the path of the root directory to monitor after the drive letter.
For example, path names should be in the form of: C:\cloverleaf\cis6.2\integrator
Or, you can click the button to open the **Select File System** dialog box, where you select the drive. Click **Apply** and the selected drive displays in the **Source** field, where you can finish specifying the path.
- 15 Click **Apply**.
- 16 Set **Source Count** to **All** to indicate when the file system approaches capacity.
- 17 Set **Comparing** to **>** and specify **90** in the adjacent text box. This triggers the alert when the disk space is more than 90% full.
- 18 Set **Duration** to **N Seconds** and specify **30** in the adjacent text box. This triggers the alert when disk space remains more than 90% full for 30 seconds.
- 19 Click **Apply**.
The next alert triggers two actions. First, a shell script runs. In this example, it is `UserCleanup`. This notifies users to clean up the directories. Then, a message is sent to root at the specified trigger.
- 20 In the Actions pane, select **exec** from the **Alert Action** menu. The pane reconfigures for command input.
- 21 Specify the name of the script (`UserCleanup`) in the **Command** text box.
- 22 Click **Update**.
- 23 Click **Apply**.
- 24 On the Alerts list of the Alert Configurator, click the alert configured, if necessary.
- 25 Click the **Append** tool in the Actions pane.
- 26 Select the **exec** alert action.
- 27 In **Command**, specify `mailx -s '%A' root < \dev\null (UNIX)`.
This sends a message to root with the "disk % full" subject line and an empty message body.
- 28 Click **Update**.
- 29 Click **Apply**. The new alert configuration displays on the Alerts List.
- 30 Click **Save** on the Alert Configurator toolbar after configuring all alerts. Or, select **File > Save As** to open a file selection dialog box.
- 31 Specify a file name, or select from the listed files and click **Save**.

Persistent alert

This alert configuration creates an alert that fires when the status of thread `conn_1` is found to be down. Then, it continues to fire again every 30 seconds until the alert is reset.

- Alert Name: `EX1`
- Alert Type: `thread status`
- Source: `conn_1`
- Source Count: `all`
- Comparing: `==` and `down`
- Duration: `once`
- Repeating: `N seconds` for `30`
- Action type: `notify`

Persistent alert with maximum repeats defined

This alert configuration is similar to the persistent alert. In addition, it contains another action message and only fires a maximum of five additional times.

- Alert Name: `EX2`
- Alert Type: `thread status`
- Source: `conn_1`
- Source Count: `all`
- Comparing: `==` and `down`
- Duration: `once`
- Repeating: `N seconds` for `30` with a Max of `5`
- Action type: `exec`, using,

```
hciguimsg -alert -message "%N has fired %R of 5  
times
```

Using NAME to identify an alert in the logical "and" type alert

In this example, the names of the inbound latency alert and the inbound message queue alerts are used to create a logical alert. This alert starts forwarding for `conn_1`.

- Alert Name: `EX3A_c1Slow`
- Alert Type: `iblat`
- Source: `conn_1`
- Source Count: `all`
- Comparing: `>5`
- Duration: `once`
- Repeating: `N seconds` for `30`
- Action type: `none`

- Alert Name: **EX3B_c1QDLimit**
- Alert Type: **ipque**
- Source: **conn_1**
- Source Count: **all**
- Comparing: **>15**
- Duration: **N seconds for 30**
- Action type: **none**
- Alert Name: **EX3C_forwardingC1**
- Alert Type: **and**
- Source: **EX3A_c1Slow EX3B_c1QDLimit**
- Source Count: **all**
- Comparing: **== and true**
- Duration: **once**
- Action type: **exec {hcicmd.pl -p process_1 -c "conn_1 fwd_start"}**

Using the NAME key to provide more information in the alert action

In this example, %N is used to provide more information to the operator when the alert is fired.

- Alert Name: **EX4A: CERN_IN down, call Support (800)555-1234**
- Alert Type: **thread status**
- Source: **CERN_IN**
- Source Count: **all**
- Comparing: **== and down**
- Duration: **N Minutes and 30**
- Action type: **tcl {notifySwitchboard "%N" "%A"}**
- Alert Name: **EX4B: RAD_OUT down, check server, call Emily (650)555-1234**
- Alert Type: **thread status**
- Source: **RAD_OUT**
- Source Count: **all**
- Comparing: **== and down**
- Duration: **N Minutes and 30**
- Action type: **tcl {notifySwitchboard "%N" "%A"}**

Preventing redundant alerts that would be caused by normal process shutdown

In this example, conn_1 thread belongs to process process1. To prevent duplicate alerts when process1 is shut down, check **Thread Only**.

- Alert Name: **EX5A**
- Alert Type: **thread status**
- Source: **conn_1**

- Thread Only: selected
- Source Count: **all**
- Comparing: == and **down**
- Duration: **once**
- Action type: **notify**
- Alert Name: **EX5B**
- Alert Type: **protocol status**
- Source: **process1**
- Source Count: **all**
- Comparing: == and **dead**
- Duration: **once**
- Action type: **notify**

Defining complementary alert windows

In this example, two alerts are set up so that email during the work week is addressed to one group of support. For the weekend, the same alert sends email to another group.

- Alert Name: **EX8D**
- Alert Type: **thread status**
- Source: **conn_1**
- Source Count: **all**
- Comparing: == and **down**
- Duration: **once**
- Time Windows: **1-5/9:00-16:59/*/***
- Action type: `tcl {notify %N %A Level1Support}`
- Alert Name: **EX8E**
- Alert Type: **thread status**
- Source: **conn_1**
- Source Count: **all**
- Comparing: == and **down**
- Duration: **once**
- Time Windows:
- Not: selected
- Action type: `tcl {notify %N %A AfterHoursSupport}1-5/9:00-16:59/*/*`

Logical "OR" alert

This example shows an alert that fires if the protocol thread is down or its outbound queue depth is greater than 25.

- Alert Name: **EX13A**
- Alert Type: **protocol status**
- Source: **ob_1**

- Source Count: **all**
- Comparing: **==** and **down**
- Duration: **once**
- Action type: **none**
- Alert Name: **EX13B**
- Alert Type: **opaque**
- Source: **ob_1**
- Source Count: **all**
- Comparing: **>25**
- Duration: **once**
- Action type: **none**
- Alert Name: **EX13C**
- Alert Type: **and**
- Source: **!EX13A !EX13B**
- Source Count: **all**
- Comparing: **==** and **false**
- Duration: **N minutes** and **30**
- Action type: **notify**

Helloworld

This alert calls helloworld every 12 polling passes, and removes the file once there are 5 lines in the file.

- Alert Name: **EX12**
- Alert Type: **tcl**
- Source: **TCL {helloworld \$env(HCISITEDIR)/helloworld.log}{TIME {count 12}}**
- Source Count: **all**
- Comparing: **>=5**
- Duration: **once**
- Action type: **{exec {rm \$env(HCISITEDIR)/helloworld.log}}**

To "bad"

This example shows how using the shortcut (To: bad) simplifies an alert definition file.

- Alert Name: **EX11A**
- Alert Type: **protocol status**
- Source: **conn_1**
- Source Count: **all**
- Comparing: **==** and **error**
- Duration: **N seconds** and **30**
- Action type: **notify**
- Alert Name: **EX11B**

- Alert Type: **protocol status**
- Source: **conn_1**
- State From: **up**
- To: **down**
- Source Count: **all**
- Comparing: **diff**
- Duration: **N seconds** and **30**
- Action type: **notify**
- Alert Name: **EX11C**
- Alert Type: **protocol status**
- Source: **conn_1**
- State From: **up**
- To: **opening**
- Source Count: **all**
- Comparing: **diff**
- Duration: **N seconds** and **30**
- Action type: **notify**
- Alert Name: **EX11D**
- Alert Type: **protocol status**
- Source: **conn_1**
- State From: **opening**
- To: **down**
- Source Count: **all**
- Comparing: **diff**
- Duration: **N seconds** and **30**
- Action type: **notify**

The equivalent alert behavior that is defined with a single "To: bad alert":

- Alert Name: **EX11_ALL**
- Alert Type: **protocol status**
- Source: **conn_1**
- To: **bad**
- Source Count: **all**
- Comparing: **diff**
- Duration: **N seconds** and **30**
- Action type: **notify**

From "up" to "ineof"

In this example, the alert arms when conn_1 transitions from "up" to "ineof."

A state transition that is from any other state in addition to "up" does not arm the alert.

A state transition from "up" to any state in addition to "ineof" does not arm the alert.

If the state stays in "ineof" for 30 seconds, then the alert fires.

- Alert Name: **EX10**
- Alert Type: **protocol status**
- Source: **conn_1**
- State From: **up**
- Source Count: **ineof**
- Comparing: **diff**
- Duration: **N seconds** and **30**
- Action type: **notify**

From "up" to anything

In this example, the alert arms when conn_1 transitions from "up" to any other state. A state transition that is from any other state in addition to "up" does not arm the alert. If it stays in that destination state for 30 seconds, then the alert fires.

- Alert Name: **EX9**
- Alert Type: **protocol status**
- Source: **conn_1**
- State From: **up**
- Source Count: **all**
- Comparing: **diff**
- Duration: **N seconds** and **30**
- Action type: **notify**

Hold based on queue depth

In this example, if the inbound message queue for ib_tcp_10111 is over 25, this set of alerts holds the flow of the client ob_tcp_10111. The flow is held until the inbound message queue for ib_tcp_10111 is below 10.

- Alert Name: **EX14A_overLimit**
- Alert Type: **ipque**
- Source: **ib_tcp_10111**
- Source Count: **all**
- Comparing: **>25**
- Duration: **once**
- Action type: **none**
- Alert Name: **EX14B_underLimit**
- Alert Type: **ipque**
- Source: **ib_tcp_10111**
- Source Count: **all**
- Comparing: **<10**
- Duration: **once**

- Action type: **none**
- Alert Name: **EX14C_holding**
- Alert Type: **hold**
- Source: **ob_tcp_10111**
- Source Count: **all**
- Comparing: **==** and **true**
- Duration: **once**
- Action type: **none**
- Alert Name: **EX14D_startHold**
- Alert Type: **and**
- Source: **EX16A_overLimit !EX14C_holding**
- Source Count: **all**
- Comparing: **==** and **true**
- Duration: **once**
- Action type: `exec {hcicmd.pl -p process_1 -c "ob_tcp_10111 phold_obd"}`
- Alert Name: **EX14E_stopHold**
- Alert Type: **and**
- Source: **EX14B_underLimit EX16C_holding**
- Source Count: **all**
- Comparing: **==** and **true**
- Duration: **once**
- Action type: `exec {hcicmd.pl -p process_1 -c "ob_tcp_10111 prls_obd"}`

Lock Manager process status alert

This example shows how the Lock Manager of site prod1 can be monitored.

- Alert Name: **EX16**
- Alert Type: **pid**
- Source: **SITE prod1** with **DAEMON hcilockmgr**
- Source Count: **all**
- Comparing: **==** and **dead**
- Action type: **notify**

Messages in error DB

This alert notifies when there are 15 or more messages from any thread in the error database.

- Alert Name: **EX1**
- Alert Type: **errdb**
- Source: **All Threads**
- Mode: **actual**
- Source Count: **all**

- Comparing: **>=15**
- Duration: **once**
- Action type: **notify**

File does not exist

This alert monitors the Lock Manager's PID file's existence.

- Alert Name: **EX18**
- Alert Type: **file size**
- Source: `$env(HCISITEDIR)/exec/hcilockmgr/pid`
- Mode: **actual**
- Source Count: **all**
- Comparing: **!=--1**
- Action type: **notify**

Email action with alert message

This sample alert sends an email with an alert message.

- Alert Name: **conn1down**
- Alert Type: **thread status**
- Source: **conn_1**
- Mode: **actual**
- Source Count: **all**
- Comparing: **==** and **down**
- Duration: **once**
- Action type: **email**
- To: **level1support@mycompany.com**
- Subject: **Cloverleaf Alert: %N**
- Message: **%A**

Advanced Email Options dialog box:

- SMTP server: **smtp.mycompany.com**
- Domain: **mycompany.com**
- User Address: **cloverleaf@mycompany.com**

Thread protocol status

This sample alerts you when a thread has lost connectivity for 10 minutes.

- Alert Name: **lostconn**
- Alert Type: **protocol status**

- Source: **his_1**
- Mode: **actual**
- Source Count: **any**
- Comparing: **==** and **opening**
- Duration: **N minutes** and **10**

Last receive

This sample alerts you when monitorD finds that the last message was received greater than or equal to 28800 seconds (8 hours) ago.

- Alert Name: **lastreceive-8hrs**
- Alert Type: **last receive**
- Source: **lab_1**
- Mode: **actual**
- Source Count: **any**
- Comparing: **>=** and **28800**
- Duration: **once**

Outbound queue depth

This sample alerts you when there are more than 100 messages in state 11 for over 5 minutes.

- Alert Name: **outbound queue depth - 100 5 mins**
- Alert Type: **outbound queue depth**
- Group: **outbound**
- Source: **lab_1 lab_2**
- Mode: **actual**
- Source Count: **any**
- Comparing: **==** and **100**
- Duration: **N minutes** and **5**

Viewing the alert log file

The **Log File Type** menu on the Site Daemons GUI is used to select an alert log for viewing.

Alert Log is only selectable when the current daemon is All Daemons or Monitor Daemon.

The `alerts.log` file records when an alert is fired and when it is cleared.

This log gives you a way to track alert firings along with actions that reset the fired alert.

Note: Alert action firings are no longer written to the `hcimonitorD.err` log. Only errors and warnings are written to the `hcimonitorD.err` log.

A sample alert file (`sample.alert`) can be found at `$HCIR00T/Alerts`.

BOX Manager

A Buildable Object eXchange (BOX) is a file package consisting of a collection of related system configuration files. Through the IDE, a BOX is transferred from one system root to another. Every BOX has a manifest file to record its content, the time of creation, the version, and descriptive notes.

BOX also provides the ability to share configuration resources between system sites on several hosts.

A BOX can include these objects:

- Thread/destination and its process definition
- Database schemas
- Tcl procs
- Java UPoCs
- PDLs
- Variants of message format
- EO configurations
- Xlate and XSLT
- Lookup tables

A BOX can also include these on the physical file level:

- NetConfig and thread/process notes files
- Database schema files and corresponding database configuration
- Tcl script files
- Java UPoC class files
- PDL files
- Format files: FRL, VRL, HRL, LDL, XML, HL7, X12, and other HMD variant files
- EO configuration files
- Translation configuration files, XSLT files, Lookup table files

An audit trail is created when creating, importing, exporting, deploying, and creating a site for a BOX. This trail is viewed from the Audit Log Viewer.

If your Java classes are packaged to a `jar` file, then you can add this file to the BOX using the GUI. Its original folder structure is remembered in the BOX `manifest.xml` file during BOX creation.

Dependencies between Tcl files

The BOX functionality does not detect dependencies between Tcl files.

For example, a TPS procedure called `mytps` is included in a Tcl file `test.tcl`. This TPS procedure calls another procedure `myother` of type `other` which is located in another Tcl file `test2.tcl`.

If you use `mytps` in a thread and you include this thread in a BOX, then `test.tcl` is added to the BOX. The `test2.tcl` file is not added, so after deployment you miss the `myother` procedure.

The BOX tool does not detect the requirement for `test2.tcl` because this is only detected by parsing the Tcl code. You should be aware of the `test2.tcl` file missing in the BOX.

To solve this, files can be manually added to a BOX.

This also applies to tables. Tables and format configurations can be referenced in a Tcl process.

BOX Manager GUI

The BOX Manager is a tool that manages BOXes of the system host, and supports:

- Listing BOXes in several hosts
- Transferring BOXes between hosts
- Viewing BOX detail information
- Removing a BOX from the BOX repository
- Importing a BOX into the current BOX repository, or exporting BOXes to the destination directory.
- Deploying a BOX to the current site, or creating a new site with the BOX.

Note: BOX Manager respects security server access control if the current host runs in advanced security mode.

In the **BOX Manager** dialog box:

- The panel lists all BOXes of the current host by default.
- BOXes are sorted by name in alphabetical order.
- Pausing the mouse on a BOX bitmap opens a tooltip showing the BOX name.
- The BOX bitmap is selectable.
- The selected BOX has a blue rectangle around it.
- Multiple BOXes can be selected.

GUI options

Option	Description
Copy/Paste	<p>You can copy and paste a BOX entry or folder entry into another folder with the same name. If you copy and paste a folder entry, then the sub-folders are also moved.</p> <p>These are supported across multiple IDEs and BOX Managers.</p> <p>The drag and drop action is supported between Local and Remote BOXes.</p>
New Folder	<p>This creates a new folder on the selected BOX view. The name is editable.</p> <p>The BOX search result view does not support this action.</p>

Option	Description
Change View	<p>Clicking this opens a menu of options:</p> <ul style="list-style-type: none"> • List is the default view. • Details is convenient for users with a large number of BOXes. Columns can be customized by right-clicking in the header area and selecting/clearing columns as required. • Tree lists BOXes as a tree, and is useful for displaying the contents of user-created folders.
Search field	<p>Search parameters can include a Name, Metadata, and Resource search.</p> <p>These options are accessed by clicking the arrow.</p>

GUI panels

In the Current Host panel, if only one BOX is selected, when you right-click the BOX a menu opens with a list of options, including:

- **Import** a BOX package to the current root
- **Export** the selected BOX as a file package to the local file system.
- **Deploy** the BOX to the current site. This menu item is enabled only when the selected BOX belongs to the Cloverleaf root to which the current connected site belongs. After a BOX is deployed, the Site Manager is automatically refreshed.

In the Remote Host panel, selecting **Expand** opens the Remote Host panel.

This panel is for connecting and displaying BOXes on another host. At the top it shows the currently connected host. The BOX icon in this panel lists all BOXes from the currently connected host.

Select Host dialog box

Selecting **Select Remote Host** opens the **Select Host** dialog box.

The **Host** menu on this dialog box contains a list of hosts to which the IDE has connected, excluding the current host. This list is editable, and you can directly specify a new host name or IP address.

Port is for specifying a remote host server port number. This is pre-populated with the port number found in `client.ini`. If no port is found, then it uses the default RMI registry port.

If you specify a host name or IP address that is pointing to the current host, then a message opens: The remote host cannot be the same as the current host.

When you click **OK**, the BOX Manager attempts to connect to the target host.

If the connection is successful, then the **Select Host** dialog box closes and the target host name/IP address is shown. The panel context is refreshed with the BOX list from the new host.

An error message opens if the connection attempt fails.

User-defined folders

To help with BOX management, you can create your own folders. User-defined folders are a convenient way to group BOXes, where you can define folders and sub-folders.

To create a user-defined folder, select **Edit > New Folder**, or right-click in the BOX panel and select **New Folder**.

Drag and drop a BOX or copy/paste it into the folder.

When you open a folder, use **Up One Level** to return to the main view.

BOX search

The BOX search feature uses a tag metadata. You can define a string as the keyword of a BOX, and a BOX with the same tag can be searched.

A default metadata tag is on the **New BOX Property** dialog box and **New BOX Basic Information** dialog box.

The BOX toolbar has a **Search** function where the menu lists the supported search types of name, metadata, and resource.

You select a type and specify the search content. The search scope is the current view folder and its sub-folders.

Search types include:

- Name search: BOX shows the search result in left panel.
- Metadata search: Use any metadata property value from the menu list. The search result list has two columns, Location and Metadata. The Metadata column shows the metadata fields and values of a BOX that match the search content.
- Resource search: Select from the menu list of types.

By default, you can perform a fuzzy query for all types of resources. For example, the menu list is left empty and **abc** is specified as the search content. Then threads, processes, and xlate files with "abc" inside their name are returned. Search results are shown in the Location and Resource columns.

If you specify **abc.hrl** as the search content, then BOXes that contain the file The Resource column lists all resources that match the search criteria. **abc.hrl** are returned. This function applies also to other types of resources whose file name has that file extension.

The same result is returned if you selects HRL configurations as the resource type and only specify **abc** as the search content.

For HMD formats, in addition to searching by resource name, you can also specify the version as search content. The BOXes that have that version HMD format are returned.

Note: Search criteria can contain special characters, such as " ", "-", ".", and so on.

BOX Property dialog box

Double-clicking a BOX icon or clicking a Properties menu item opens the **BOX Property** dialog box.

The **BOX Property** dialog box resource tree shows several level resources in different root nodes for grouping. For example, Site, Master Site, and Root. This is similar to the BOX resource trees in the Create BOX and Create Site with BOX wizards.

On the **Resources** tab, you can click **Add Resources** to add Tcl procs.

Click the arrow to open a menu where you can to add resources, for example, database schema or X12 configurations.

BOX Property Overview tab

In the **Overview** tab, **Name** is required, and must be unique for the host. Lowercase, uppercase, underscore, and dot are supported.

The Metadata table is for specifying additional BOX properties. By default, this table lists default properties. You can modify both properties and values in the table cells.

New appends a new row to the table for a new property.

Remove deletes one or more added properties. Empty property rows are ignored when saving.

BOX Property Resources tab

The tree view lists all resources belonging to this BOX.

When a resource is highlighted, the **Text** tab in the Description area is for adding descriptions of the currently selected tree node. The **HTML Preview** tab shows the description in HTML format. This tab is not editable.

Thread, destination, and process descriptions are saved directly as notes. Other resource descriptions are saved into the BOX manifest file.

On this tab, you can:

- Select and add new resources to BOX
- Remove existing resources from BOX
- Rename BOX resources
- Configure destination/thread environment properties

Functions include:

Add Resources opens a menu for selecting and adding all types of BOX resources. Click the folder button to add Tcl procs. Click the arrowhead to add resources, for example, database schema or X12 configurations.

Delete Resources deletes the currently selected node on the resource tree. These actions take effect when **OK** is clicked.

When you select a resource node from the resource tree, the selected resource name and description populates **Resource Name** and **Resource Description**.

When the resource type is Database Schema, the right panel contains an additional **Configuration** tab.

On this tab, you can reconfigure **Database URL**, **User Name**, **Password**, and **Schema**. Modifications are saved in `dbconfiguration.ini`.

When the resource is a thread or destination, the **Configuration** tab displays, where you can configure environment properties. Configuration parameters are the same as the **Confirm Environment Configurations** dialog box of the Deploy BOX to Site wizard.

Configuration results are saved into the `NetConfig` file of BOX.

Adding global variables to BOX

You can add global variables to BOX when there is only a global variable used in TCL.

For example, you must add values to a Tcl proc when only global variables are in the proc. You also must reference a global variable in a Tcl proc.

To do this, manually add global variables that are not automatically detected.

On the **BOX Property > > Resources** tab, there is a "+" icon for adding a global variable. This opens the **Add Global Variable** dialog.

You can have global variables in a Tcl proc and not `NetConfig`.

When global variables are already in the tree, this option is not visible.

The BOX Manager supports adding current and master site global variables when no related global variable `ini` file is in the BOX.

- When the current and master site global variable `ini` file exists in a selected BOX, the "+" option (Add Global Variables) does not display on the **Resources** tab.
- When the current global variable `ini` file exists in a selected BOX, clicking "+" opens a menu listing the available master site global variable files.
- When the master site global variable `ini` file exists in a selected BOX, clicking "+" opens a menu listing the available current site global variable files.
- When no global variable `ini` file exists in a selected BOX, the "+" displays, where you can add current and master site global variable files. All global variables in the global variable's `ini` file are added.

BOX Property Configuration Files tab

On this tab you can configure environment-related thread properties as a template, and reuse these configurations for several deployments of the same BOX.

The left panel lists the existing configuration files. Select a file name to see its corresponding configurations on the right panel.

The right panel shows the selected file's configuration. The configuration file name is shown in **Name**. This can be renamed.

New configuration creates a new configuration file. Selecting a file populates the right pane with the BOX default configuration.

Replicate the selected configuration creates a new configuration file with the same values as the currently selected configuration file. Selecting this adds a new configuration file name to the file list.

Apply saves the editing results to the configuration file. If a file name is modified, then this action changes the file name on the system as well.

Save As is used when you edit a BOX and must keep the original BOX unchanged, saving the current editing results into a new BOX. This option is also available when you specify a new name on the **Overview** tab.

Creating a new BOX

From the NetConfig tool, you select threads or destinations to create a BOX. All referenced files are automatically included.

During BOX creation, configurations are defined as parameters that are editable. BOX resources can be edited and thread environment properties can be configured similar to the BOX deployment Wizard. This helps when creating standard interfaces and BOXes for sharing and deploying to other environments.

- 1 Select the threads/destinations to include in the BOX.
- 2 Right-click and select **Create BOX**. You can also select **File > Create BOX**.

After selecting **Create BOX**, the system checks if there are any validation errors for the thread configurations of the current NetConfig. If so, then a message opens prompting confirmation to continue. The error message differentiates between included threads and threads not included, so you can make an informed decision.

- 3 Selecting **Create BOX** opens the New BOX wizard.

On this dialog box you configure the BOX basic information. Existing properties can be modified or new properties added on this dialog box. Similar to the **BOX Property** dialog box, the **Name** field is required, and cells on the Metadata table are editable. **Version** and **Create Time** are auto-populated.

The BOX **Name** supports only letters, digits, underscores, or dots in the middle, and can be up to a length of 100.

For **Location**, specify the location or click **Browse** to open a file browser where you select the location. The top folder is `$HCIRoot\box`. If the location does not exist, then when you click **Finish** after configuration a confirmation opens for users who require creating the directory. The generated BOX entry is stored into the specified location.

Metadata values are saved in a `box.ini` file located at `\integrator\box`. This removes the necessity of re-entering common values. For example, Author and Organization are often going to be the same value. Having these values saved in an `ini` file saves you the time of re-entering those values over again.

The `box.ini` file is a template for BOX properties. Values that are defined in this file are used as the default properties for the next BOX creation. Selecting **Save As Default** saves the current properties and values to `box.ini`.

Selecting **Restore** restores all values to the system default values. This feature does not affect `box.ini`. When **Restore** is clicked, the list table is refreshed with the system's built-in properties, not the properties that are defined in `box.ini`. This is useful when it is not necessary to use the template properties, to reconfigure from scratch. Values that are saved as default are still loaded when creating a new BOX.

Properties include:

- **Author:** Current user name. In basic/advanced security, it is the log-in user. In Windows, it is the log-in user name when in non-security mode.
- **Create Time:** The BOX creation time
- **Organization:** Organization name
- **Source Data Type:** Data format type
- **Source System:** System name
- **Source System Version:** System version
- **Tag:** BOX with same tag can be searched through the BOX searching function
- **Target Data Type:** Data format type
- **Target System:** System name
- **Target System Version:** System version
- **Version:** BOX version, starting from 1.0

4 Click **Next**. This opens the **Options for References Resources** dialog box.

On this dialog box, you can include or skip referenced resources.

- Some referenced threads/destinations are not selected on NetConfig.
For example, there are three threads named "a", "b" and "c." Thread "a" routes to "b" and "b" routes to "c." If only thread "a" and "b" are selected to create a new BOX, then the new BOX is incomplete without thread "c."
If you do not include that thread, then all the routes leading to that thread are not added into the new BOX; otherwise, all referenced threads are added into the BOX. If these referenced threads also have routes to other threads, then other threads are also added into the new BOX.
- Some referenced resources are from master site or the Cloverleaf root.
If you specify package resources from anywhere, then resources from the master site or root level are packaged along with site level resources. When deploying this BOX in the future, there is no distinguishing of the master site level, root level, or site level resources; they are all at the same level. In the Master Site or Root Resources section, options are grouped into two categories.
 - Whether to include Master Site or Root level resources for reference finding:
 - Ignore non-current site resources:** Any resources from a Master Site or Root are not returned when reference processing. They are not displayed on the BOX resource tree in the next step of the Create BOX wizard.
 - Include non-current site resources:** Master Site or Root level resource files are returned when references processing, and these resources are included in the new BOX.
 - How to package Master Site or Root level resources:
 - Replicate and package as current site resources:** New BOX resource files are organized as before, where Master Site or Root level resource files are put into site level resource folders.
 - Replicate and package as original location:** Master Site and Root level resource files are saved separately from site level resource files. In this way, other level resources return to their original folders during BOX deployment.
- In Thread Route Deployment, when **Only keep the routes with destination threads in BOX** is selected, the route messages and route replies are deleted when the destination thread is not packaged in the BOX.

- 5 Click **Next**. This opens the **Modify Resource Description** dialog box. On this dialog box, all required Cloverleaf artifacts which are referenced by previously selected threads are listed on the left-hand side. They are categorized by type as a tree structure.

The BOX resource tree shows other level resources in other root tree nodes:

- Site level resources are grouped into a root node named Site.
- Master Site level resources are grouped into a root node named Master Site.
- Root level resources are grouped into a root node named Root.

Depending on the selected option combination from the previous dialog box, this tree only shows site level resources or all level resources.

The right-hand side text area is for entering further descriptions of any packaged item.

The IDE auto populates all referencing resources of the selected threads/destinations, and optionally includes those resources located in the master site and root. All included resources in the BOX are the same level regardless of whether they originate from the master site or root.

For process/thread/destination nodes, the process/thread/destination notes are shown in the text area by default. Modified descriptions for process/thread/destination are not saved in the manifest file, but propagated into the packaged process/thread/destination notes. For other nodes, the description text area is initially blank. Updated descriptions are saved to the manifest file.

The BOX resource tree is editable. The toolbar's **Add Resource** option is used to manually add most types of BOX resources. Clicking the arrow next to **Add Resource** opens a listing resources. Selecting **Add Libraries** opens the **Add Additional Libraries** dialog box.

- 6 Because BOX supports to remember a file's original path, the Java UPoC libraries path is also remembered, similar to the common library files. There is no requirement to distinguish Java UPoC library files from common library files.

Selecting **Add File** or **Add Folder** opens a file chooser, where you can select files/folders under the hci ROOT folder.

Click **OK** to add the selected library resource files to the BOX resource tree. This is where library files from other levels are grouped under other tree nodes.

- 7 When a file is selected, its name and description display in the **Resource Name** and **Resource Description** fields. These can be changed as required. This is useful for users that must rename a thread or process to match production during BOX creation.

The resource name is validated by the system and prompts an error message if an invalid name is entered.

- 8 You can also manually add/delete procs or other files on the tree.

For example, suppose you have written a TPS procedure called `mytps` which is included in a new Tcl file `test.tcl`. This TPS procedures calls another procedure `myother` of type `other` which is located in another Tcl file `test2.tcl`. If you use `mytps` in a thread and you include this thread in a BOX, then `test.tcl` is added to the BOX. The `test2.tcl` file is not added. After deployment, you miss the `myother` procedure. The BOX tool does not detect the requirement for `test2.tcl`. This is only detected by parsing the Tcl code. As a user, you should be aware of the `test2.tcl` file missing in the BOX. The possibility to add files manually in a BOX solves this issue.

- Select **Add Resources** to open the **Add Procs** dialog box, from which you can manually add procs to a BOX. Adding a selection to the Selected list and then clicking OK adds it to the **BOX** dialog box.
- Click the **Add Resources** arrow to open a list of available resources which you can manually add to a BOX.

After a selection is made, the appropriate dialog box opens showing the available configurations, scripts, and so on. Adding a selection to the Selected list and then clicking **OK** adds it to the **BOX** dialog box.

- 9 If a Database Schema resource node is selected, then a **Configuration** tab displays on the right side of the dialog box.

Modifications to the configuration fields are saved into the `dbconfiguration.ini` file of the BOX being created.

- 10 Click **Next**. Set the environment configurations of the BOX being created. This assists in sharing and deploying to other environments when distributors create standard interfaces and BOXes.

The creator of the BOX knows the configuration values to set, so the receiver of the BOX has no requirement to do any configuration.

This configuration is the same as the **BOX Property** dialog box.

- 11 Click **Finish**. A new BOX is created with the given name under the folder `CL_ROOT/box`.

Example: Creating a BOX for collaboration

User1 has developed an interface between VendorA and VendorB and must share this with their peers. To do this:

- 1 User1 opens Network Configurator and selects the two threads that are used in the interface.
- 2 User1 right-clicks and selects **Create Box** from the context menu.
- 3 On the dialog box, User1 names the BOX "User1_Box" and adds a description of "Interface created by User1."
- 4 User1 needs to include master site and root level resources. After clicking **Next**, User1 selects **Include non-current site resources**.
- 5 To share this BOX with users that may not have master site enabled, User1 selects **Replicate** and package as current site resources.
- 6 User1 clicks **Next** to go to the next page. User1 has no additional files to add to the BOX, and clicks **Finish**. The BOX is created.

Now, User1 needs to export this to a BOX file for sharing.

- 1 When User1 opens the BOX Manager, User1 sees "User1_Box" in the list and selects it. The BOX is created. Now User1 needs to export this to a BOX file for sharing.
- 2 User1 specifies the destination directory `C:\TEMP`.
- 3 User1 does not assign a password to the BOX.
- 4 User1 clicks **OK**. The `User1_Box.box` file is created in `C:\TEMP` for User1 to share with others.

BOX refresh

The BOX **Refresh** function refreshes everything within a BOX against the site/mastersite/root. If some resource files or artifacts have been changed with new references, then these newly referenced resources are also packaged into the BOX.

BOX records the information about where it was created. It also records the user-selected options when the BOX was created with the Create BOX Wizard or BOX command.

For example, “dealing with references” and “dealing with mastersite and root-level resources” properties are required to refresh the BOX with the correct context. For this, BOX records the host ID, host version, site/mastersite name, thread names, and user options into a BOX `manifest.xml` file.

For the instance name, multiple installations to one machine are supported, where every install has a unique name.

Existing BOXes can be refreshed, instead of having to be recreated. BOX searches for any changes, and present them for review.

During refresh:

- Refresh checks for newer versions of those files already included.
- When BOX resources are found that do not exist in the target environment, one of these options is taken to proceed:
 - Prompt the user with the exception and stop the refresh.
 - Ignore this and continue to refresh other resources.
 - Remove the resource from BOX since it has been removed from the target environment.
- When new referenced resources are found that do not exist in the current BOX, one of these options is taken to proceed:
 - Add new referenced resources into BOX (default choice).
 - Discard new referenced resources.

Right-clicking a BOX opens a context menu that contains **Refresh**. Selecting this opens a dialog box with refresh options. For example, options for what to do if resources do not exist on the target site, and how to deal with new found resources.

- **Refresh** checks for newer versions of those files already included.
- New references are still subject to the limitations of BOX creation.

Clicking **Next** gives you a preview of the new BOX. You can filter the view to show only the differences, or everything.

An error is given if a BOX is checked out by other users.

Resource tree nodes have indicators on their status:

- A resource with a plus-sign (+) indicates this is a new resource that is going to be packaged in the current BOX. A “To be added” tooltip is shown when the pointer pauses here.
- A resource with an "x" indicates the resource does not exist in the current environment and is removed. A “To be removed” tooltip is shown when the pointer pauses here.
- A resource with an arrow indicates the resource in BOX is different from the corresponding resource in the current environment and is updated. A “To be updated” tooltip is shown when the pointer pauses here.
- Node names of all changed resources are highlighted in red to make the changes more visible.

The filtering function displays only nodes with changes.

Command line interface

The BOX command line supports the `-r` option for BOX refresh.

```
hcibox {-i filepath | -e filepath | -d sitename |
-c sitename | -s sitename | -n sitename
thread[,thread...] [[r][m][p]] | -r [[i|s|r][a|d]]} BOX
[-f] | [-v] | [-h | -help] | [-x cfgfile=configurationfilename]
```

`-r [[i|s|r][a|d]]` refreshes the BOX.

- `[i|s|r]` indicates that when refreshing a BOX, what to do if some BOX resources do not exist on the target environment.
 - `i` ignores the resource and continues to refresh others (default).
 - `s` stops the refresh and prompts an exception.
 - `r` removes the resource from BOX.
- `[a|d]` indicates how to deal with new referenced resources found in site/mastersite/root.
 - `a` adds new referenced resources into BOX (default).
 - `d` discards the new referenced resources.

Example: Refreshing a previously created BOX

User 1 has created a BOX containing a Cloverleaf interface. Changes are made to the interface using the IDE. User 1 must refresh the BOX that was created earlier instead of creating a new BOX.

To do this:

- 1 User 1 opens the BOX Manager in the site where the Cloverleaf interface was created that is referenced by the BOX.
- 2 User 1 sees "User1_BOX" in the list and right-clicks the BOX. This opens a context menu containing the **Refresh** option.
- 3 User 1 clicks **Refresh** and a dialog box opens containing a list of all the changes that are about to be made.
- 4 User 1 clicks **Continue** to perform the BOX refresh. The new configuration objects are updated in the BOX.

How to BOX java driver plug-in threads

The plug-in java-protocol thread frequently contains configuration files. For example, for CAA-WS, there are `%HCISITE%/javadrivier/applicationContext_<threadName>.xml` files.

These files should be automatically packaged into the BOX. The BOX does not know if a plug-in java-protocol thread has specific files. It cannot automatically locate and package these files when creating a BOX.

To do this manually, select the files as library files when creating a BOX for such a thread.

Exporting BOX as a file package

The BOX package is a zip package. The GUI supports exporting multiple BOXes at one time.

You can optionally add password encryption to use when zipping/unzipping a BOX package during import/export.

- 1 From the BOX Manager, right-click a BOX and select **Export**.
- 2 Click **Browse** to open a local directory chooser for specifying the destination folder where the exported BOX file package should be located.
- 3 Clicking **OK** on the browser compresses the BOX file with the name \$BOX_NAME\$.box and saves it under the selected destination directory.

Importing a BOX package to the current host

A BOX package can be exported in the compressed format and later imported. From the BOX Manager, you can import the BOX file into the BOX repository.

Note: If you are importing a BOX from a Windows host server to a UNIX host server, then the Windows CRLF configuration files must be converted to LF format. The UNIX engine does not support CRLF files. To do this, use `dos2unix`.

- 1 Right-click in the BOX Manager to open a menu where you can select **Import**.
- 2 On the **Import BOX** dialog box, click **Browse** to open a file browser. Different platforms might have different file browser layouts. The supported file type is *.box.
If the BOX being imported has been password encrypted, then specify the **Password**.
- 3 After locating the BOX file package, click **OK**. This takes you back to the **Import BOX** dialog box. Then click **OK** on the **Import BOX** dialog box. The IDE checks if the selected file is a valid BOX package. If it is, then this BOX is imported into the BOX repository of the current host. The BOX Manager is refreshed to show the imported BOX after the **Import BOX** dialog box is closed.
If the specified file is not a valid BOX package, then an error message opens.

Transferring a BOX between hosts

The primary purpose of BOX is for migrating configurations between various hosts and sites. Using the BOX Manager, you can make transfers between hosts.

The BOX Manager has two panels to list BOXes, one for the current host and another for connecting to another host.

Using the copy/paste function, BOXes are transferred between hosts.

- 1 On the left-hand panel which list all local BOXes, right-click the BOX to be transferred to open the menu.
- 2 Select **Copy**. Then, right-click in the right-hand panel to open the menu and select **Paste**.
You can also drag and drop to transfer BOXes between hosts.

- 3 **APaste BOX** dialog box opens, showing you the progress of the transfer.
Clicking **Cancel** interrupts the transmission and removes the partially pasted BOX items.
If the transfer is interrupted, for example, due to a weak network, then a dialog box opens to report the details.
- 4 When the transfer is complete, the BOX Manager shows the transferred BOX on the right-hand panel.
If a BOX with the same name already exists in the right-hand panel, then a dialog box opens. This gives you the option to change the name or overwrite.
 - Selecting **Paste with another name** transfers BOX to the target host with the new name.
 - Selecting **Overwrite** replaces the existing BOX with the pasted one.
 In the event that you copy and paste a BOX to the same host and select **Overwrite**, the IDE cancels the paste operation.

Deploying a BOX into the current site

BOX deployment means merging BOX resources into a site and extending the system site's configuration.

The Deploy BOX wizard respects the existing file structure of a BOX, deploying files to the levels according to the BOX file level settings.

- If a BOX contains several level files, then these files are deployed to Root/MasterSite/Site directories respectively.
 - The BOX resource file folder structure saves root/master site level files separately from site level resources.
 - If a BOX contains only site level files, then these files are deployed only to the current site directory.
 - The BOX `manifest.xml` file keeps the level to which the BOX resource belongs and saves the resource file relative location.
 - For library files, their original folder structure is maintained in the BOX `manifest.xml` file during BOX creation. The folder structure is re-created and the library files are copied into them during BOX deployment.
- 1 Right-click the BOX and select **Deploy** on the menu.
If there are any open files in the IDE that must be updated or merged, then a message dialog box opens. This informs you to close all files for the listed tools. Close the listed files and click **Deploy** again on the BOX Manager right-click menu. This opens the Deploy BOX wizard opens to BOX Summary and Resources.
 - 2 Review the tab information.
 - The **Summary** tab shows all information defined in the BOX manifest file.
 - The **Resources** tab shows a list of resources in the BOX.
 - If the BOX contains a database connection, then an additional **Configuration** tab displays, showing the current database configuration. If this needs to be edited, then a Confirm Database Configurations page is available after confirming the environment configurations.
 - 3 Click **Next**. This opens the **Merge BOX Resources into Current Site** dialog box.
This shows any site, master site, and root level file conflicts.
Note: If there are no conflicts, then the Conflict column is hidden. The BOX tree is still kept for you to see the contents.
For additional information, go to [Resolving conflicts](#).

Next is enabled when all conflicts are resolved.

- 4 Click **Next** to open the **Confirm Environment Configurations** dialog box.
This tab lists all ports for each thread/destination of NetConfig. There are usually several ports when a thread runs in another host.
- 5 You can also specify a file in which to save the configuration during the deployment. Open the **Choose Configuration File** to select a file, or click **Save** to open a dialog box for adding a new file. Files are saved in the `config` folder under the BOX. For example, `integrator\box\conn_1_tcpip\config`.
- 6 The Other Environment Configurations tab shows all system environment-related variables for threads. Verify the information on this tab. These environment-related properties, defined in NetConfig, must be modified to match the new host environment; otherwise, the process/thread might not run.
- 7 Click **Next**. If the BOX contains a database connection, then the next wizard page is Confirm Database Configurations, where you can make changes as required.
You can reconfigure **Database URL**, **User Name**, **Password**, and **Schema**. Modifications are saved in `dbconfiguration.ini`.
For example, a BOX that was created from a test environment is deployed to a production environment. The database connection might require reconfiguration to the production database.
- 8 Click **Finish**. The wizard checks file locks for all to-be-updated resources of the current site. If some files are locked by other clients, then a **Lock Conflict** dialog box opens.
 - Clicking **Yes** breaks all listed file locks. The wizard continues to update or merge BOX resources into the current site and then close itself after the operation is finished. All locks are released after deployment.
 - Clicking **No** closes the **Lock Conflict** dialog box. This returns you to the **Confirm Other Environment Configurations** dialog box. In this dialog box, you can navigate to the other wizard dialog boxes to stop or retry to break the lock.

Note: For NetConfig, new threads from the deployed BOX are located at the bottom of the dialog box. This avoids overlapping with existing threads on the NetConfig tool.

Notes when deploying a BOX

The NetConfig view file is saved when NetConfig files are added to a BOX. This removes the necessity of rearranging threads after a deployment.

When a BOX is deployed, the Site Manager is automatically refreshed. When Tcl procs are deployed to an existing site, the `tclIndex` is also updated.

It is common to include external files in a BOX, for example, cron scripts and SQLite databases. When a BOX is deployed, these folders and files are placed in their original location.

For example, if the files are imported from `$HCISITEDIR/data/db/`, then a directory path would be created during the deployment and the files placed there.

Deploying a BOX to an existing project

On the CLWizard, you can deploy a BOX to an existing site. A confirmation dialog box summarizes the items that are overwritten.

To do this, use the **Deploy** button on the toolbar. The default status is disabled. When a BOX is selected, the status changes to enabled". Clicking the button navigates to the **Deploy** page.

An existing project can be updated by deploying a BOX that has uploaded to the server.

The deploy method only supports overwriting conflict resources. A confirmation dialog box displays that lists what is overwritten.

On the user interface, you can only overwrite all conflicts. The API supports ignoring conflicting objects.

To deploy a BOX to an existing site:

- 1 Select a BOX to enable **Deploy**. This is located on the toolbar. This is located on the **BOX(Template)** tab of the CLWizard homepage. The default status is disabled. The status is enabled after you select a BOX on the tab.
- 2 Click **Deploy**. This goes to a new **BOX Deploy Wizard** page.
The wizard for deploying a BOX consists of two steps. The first step is selecting a project (site) to deploy this BOX. The second step is resolving conflicts.
- 3 Select the project to deploy. After a project is selected, **Next** is enabled.
- 4 Click **Next**. Any resource conflicts on the BOX resource tree are displayed. You can overwrite or ignore any of the conflicts.
- 5 The wizard has three buttons: **Previous**, **Next**, and **Finish**. The default status is disabled for all buttons. The first step is to select a deployed project. In this step, there is a `Search Result` label and a **Search Field** component. The default value is null.
After you enter the search keywords in **Search Field**, the matched site names displays in the result list. When a site is selected, a `Search Result` label displays the selected site name. After this, **Next** is enabled. Click this to go to the next step.
- 6 In this step, you can deploy resource conflicts. A BOX Resource tree is displayed on the page. If the resource in this BOX conflicts with the specified site that was specified in Step 1, then it displays in red.
These buttons are available to control the tree display:
 - **Show All / Only Show Conflict Resources**
 - **Expand All**
 - **Collapse All**
 To remind you that only overwrite operations are supported, there is a **I accept to overwrite all conflicts when deploying** check box below the BOX tree. **Finish** is enabled when after the check box is selected. When the result is successful, it goes to the **Templates(BOXes)** page.
If the result fails, then it stays on the current page.
- 7 In this step, there are two wizard buttons. The **Next** button is always disabled since this is the last step in this wizard. **Previous** is always enabled. Click **Previous** to select another site to deploy. After the selected site is changed, the conflict resources on the BOX resource tree are recalculate.

- 8 Click **Finish**. A busy indicator displays with the message "Deploying" until the deployment is completed. The result is displayed as a message at the left top of the page. For example, The message content is similar to BOX deployed successfully to Site. If there are any errors, then an error message displays.

Merge BOX Resources into Current Site wizard page

This wizard page shows any site, master site, and root level file conflicts.

The Site, Master Site, and Root nodes are used to group different level resource nodes.

Regardless of level, for nodes that have conflicts, you can select to rename/overwrite/ignore/merge to resolve the conflict.

Click the conflict indicator (question mark) to open the **Resolve Conflict** dialog box. If the current resource belongs to the master site or root level, then the Compare table's third column title displays as site, master site, or root. This reflects the level information.

After deploying BOX resources to all levels, Tcl procedure indexes of all levels are automatically updated.

Database protocols

When a database protocol thread is selected to create a BOX, the related database connection properties and database driver configuration are packaged. When that BOX is deployed, the database connection properties and driver configurations are also deployed.

Changes are made after deployment. For example, when you change the database connection URL from "test" to "protocol," change the database connection properties after that BOX is deployed.

Resolving conflicts

To resolve a conflict, click the conflict indicator to open the **Resolve Conflict** dialog box.

A conflict happens when there are issues merging the left-hand side resources into the site resource repository. Conflicts are in red, and a clickable icon displays in the middle column.

For example, on the left side there is a thread named conn_8, and on right side there is also a thread named conn_8. Thread conn_8 cannot be added to the current site because of a conflict in the name. To add the thread, you must first rename or overwrite it.

Points to remember when resolving conflicts:

- The Compare pane shows the results of conflict resources. Different resources have different check points.
- The **Merge** option ensures that in conflicts no routing information is lost when deploying a BOX. This option is for thread conflicts. When merging resources, except for route information, all configuration items in one thread are overwritten.
- When resolving process conflicts, the overwrite option label changes to **Overwrite Properties**.

- Selecting **Ignore** skips the resource during deployment.

Comparing resource files

File Compare on the **Resolve Conflict** dialog box launches a third-party binary to manually compare resource files.

- 1 On the **Client Preferences > External Command** tab, click the **External File Compare Command** folder button to open a file browser to locate the binary.

After locating a file compare binary, click **Open**. This closes the browser and fills **External File Compare Command** with the selected file path and the two placeholders.

You can also manually specify the full path and arguments for the binary in the field. For example:

```
C:\diff.exe @F1@F2 /i /p
```

When the command line is run, the @F1 placeholder is substituted with the BOX resource file path. The @F2 placeholder is substituted with the resource file path of the current site.

- 2 In the **Merge BOX Resources into Current Site** page of the wizard, click the conflict indicator to open the **Resolve Conflict** dialog box.

Choices for resolving this conflict are to rename, overwrite, ignore or merge.

- 3 **File Compare** in the **Resolve Conflict** dialog box is enabled for these type of resources when **External File Compare Command** is configured. Resources include:

- EO Alias
- FRL
- HRL
- Lookup Tables
- NetConfig
- Tcl
- VRL
- Xlate

Resources with these types all have one separate configuration file that can be written to a temp file and compared. For processes or threads in NetConfig, the thread or process definition segment is saved into a temp file and compared.

- 4 Clicking **File Compare**:

- Downloads the target resource files to a temp folder on the local machine.
- Replaces the two tokens of the External File Compare Command with the temporary resource file path.
- Invokes the command to launch the File Compare tool.

The temporary files are removed when closing the Deploy BOX wizard.

Resolve Conflict dialog box

Click the conflict indicator (question mark) to open the **Resolve Conflict** dialog box. If the current resource belongs to the master site or root level, then the third column title of the Compare table shows as site, master site, or root. This reflects the level information.

After deploying BOX resources to all levels, Tcl procedure indexes of all levels are automatically updated.

Overwriting the resource

If you overwrite the existing thread with the new one, then click the conflict indicator to open the **Resolve Conflict** dialog box.

Selecting **Overwrite** (**Overwrite Properties** for process conflicts) turns the font blue, indicating an overwrite.

You can overwrite individual conflicts, or click **Overwrite all Conflicts** on the toolbar to overwrite all listed conflicts.

A single arrow icon indicates that only that particular conflict is overwritten.

Ignoring conflicts

Resource conflicts can be ignored. You can apply this to an individual conflict or click **Ignore All Conflicts** from the "Merge BOX Resources into Current Site" toolbar. When **Ignore All Conflicts** is selected, all conflicts are ignored, meaning the deployment is skipped.

When all conflicts are ignored, and **Finish** is clicked, an Information dialog box opens informing you There is nothing deployed.

Renaming the resource

Resting the pointer over the conflict indicator opens a tooltip of available options.

For a name conflict, double-clicking the left-hand resource node replaces the name with a text field.

You can specify a new thread name in the text field. After entering a name, press **Enter** or click outside the text field and the text field is replaced with the new name. At this point, the new name is verified. If the new name does not pass validation, then the conflict indicator icon stays active.

The rename applies only on import time, and does not change the resource in the BOX.

All references to this thread are updated with the new name.

You can also click the conflict indicator to open a dialog box from which to rename the thread.

Port selection

A **Select** option after a port field indicates that this is a server side port. Clicking this opens the **Find Port** dialog box, which generates the next available port.

List after a port field indicates that this is a client port. Clicking **List** opens the **Ports List** dialog box, from which you can select from the most frequently used ports.

For each thread on the **Confirm Ports** dialog box, there is a field for the inter-site routing port.

For a destination, there is a field for the destination port.

The **Port** field varies according to the different thread protocols.

Environment-related settings

These settings show system environment-related variables for threads. These properties are defined in the Network Configurator, and must be modified to match the new host environment; otherwise, the process/thread might not run.

For example, for an FTP protocol thread, the FTP Server and Login Account settings are environment-based. To run this thread in another environment with a new FTP server, the FTP-related settings must first be modified.

Creating a new site with an existing BOX

- 1 Right-click a BOX and select **Create Site** from the menu.

The site name must contain only lowercase letters, digits, or underscores. It cannot be the same as an existing site name of the current host.

- 2 After entering a unique site name, click **Next** to open the **BOX Summary and Resources** dialog box.

- The **Summary** tab shows a summary of the BOX.
- The **Resources** tab shows the current resources for the BOX.
- The BOX resource tree groups different level resources into different tree root nodes: Site, Master Site, and Root.

- 3 Click **Next**.

A BOX can contain master site or root level resources. If there are files with the same names on the master site or root level folders of the current host, then these conflicts must be resolved.

- 4 Click **Next**.

This opens the **Merge Box Resource into Master Site or Root** dialog box. This is similar to the **Resolve Conflict** dialog box of the BOX deployment wizard.

The difference is that the Create Site wizard only handles master site or root level conflicts. The Deploy BOX wizard handles all level resource conflicts.

When you are finished creating the new site and deploying BOX resources to all level folders, TCL procedure indexes of all levels are automatically updated.

5 Click **Next.**

This opens the **Confirm Environment Configurations** dialog box. This dialog box is similar to the **Confirm Environment Configurations** dialog box in the Deploy BOX wizard. It lists all ports and environment-related variables for each thread of NetConfig. Ports are usually different when a thread runs in another host. This gives you the convenience of setting ports from one GUI.

6 Select the **Other Environment Configurations tab.**

To be deployed, a BOX can come from other hosts in different environments. This tab shows all system environment-related variables for threads.

7 If the BOX being created contains one or more database schema resources, then clicking **Next opens the **Confirm Database Configurations** dialog box. Verify all entries are correct or make modifications as required.****8 Click **Finish**.**

The wizard creates a new site for the current host to which the BOX belongs, and then deploys BOX resources to the new site.

If an error happens during site creation or resource deployment, then an error message opens. At this point, you are returned to the wizard to correct the error.

BOX folder structure

The BOX files are located under `CL_ROOT/box`, and the folder structure is similar to a site.

BOX data is stored in a directory hierarchy until exported to a BOX file. This hierarchy contains `_mastsite` and `_root` directories to store Master Site and Root resources.

A `manifest.xml` file is stored at the top level directory in the BOX, which contains metadata about the BOX.

This file is created when a BOX is created, and stores the information of all BOX resources. The **BOX Configuration** dialog box and other BOX operation GUIs read this file to display the BOX resource summary.

This XML file contains these parts:

- BOX title and description.
- BOX other properties. Users can add any property to this list.
- BOX resources. Resources are in various categories. Every resource node has a `name` and `type` attribute, and some types of resources might also have `version` and `parent` attributes.
- Each resource node has an attribute named `level` that indicates to which level a resource belongs.

Attributes

Attributes for the BOX folder structure include:

- `name`
The attribute name, which can be a file name, variant name, or other.
- `type`

The resource type. Different types of resources have a unique type. Valid resource attribute types include:

- `dbscheme`
- `edifact`
- `eoalias`
- `frl`
- `hl7`
- `hprim`
- `hrl-file`
- `ldl`
- `ncpdp-fb`
- `ncpdp-script`
- `ncpdb-telecom`
- `process`
- `thread`
- `destination`
- `pdl`
- `table`
- `tcl`
- `tcl-proc`
- `vrl`
- `x12`
- `xlt`
- `xml-package`
- `xml-scheme`
- `level`

Indicates to which level a resource belongs. Three strings are permitted:

 - `root`
 - `mastersite`
 - `site`
- `version`

The resource version. This attribute is optional and only for variant type resources such as hl7,edifact, and so on.
- `parent`

The parent resource name. These types of resources support this attribute:

 - `tcl-proc`
 - `thread`
 - `xml-scheme`

Saved resources

All resource files and folders of the site level are saved to the root folder for backward compatibility. For example, `\integrator\box\BOX name`.

Root and master site resources are saved in these sub-folders within each BOX folder:

- `_root`
- `_mastersite`

Command line usage

The `hcibox` command automates BOX export, import, and deployment.

The corresponding binary is found under `%HCIR00T%/clgui/bin` and is only for host server installations.

See [hcibox](#).

Restoring a site after deploying a BOX

When deploying a BOX to the current site, Box Manager auto backs-up the site resource files/folders that are overwritten to `$HCIR00T/box/backup/sitename/boxname-time` .

The original file/folder structure is retained.

Sometimes, though, it may be necessary to restore a site after deploying a BOX. For example, a deployment is finished by accident, or a problem is discovered after the deployment. These must be rolled back to resolve the issue.

- 1 Check if there are any folders under `$HCIR00T/box/backup/sitename/boxname` .
If so, then there should be a file named `database.ini`. This file is a backup of `$HCIR00T/server/database.ini`.
Copy this file from the backup directory to `$HCIR00T/server`.
- 2 Copy all other files and folders to site directory, overwriting existing files.
- 3 Check if there are any Tcl files in `BACKUP_DIR/tclprocs`.
If so, then run the `mktclindex` command to re-generate the Tcl procedure indexes. For example:

```
>setroot  
>mktclindex C:\cloverleaf\cis6.2\integrator\helloworld\tclprocs
```

- 4 Check if there is an XML package in `BACKUP_DIR/formats/xml`.
If so, then open the IDE, and on the XML Package Manager, compile the DTD/XSD files in the corresponding packages.

Database Schema Configurator

You can use this tool to configure database connections in the Network Configurator and map database schemas in the Translation Configurator. With these supports, you can significantly reduce Tcl script writing and improve usability on database-related configurations.

Before using the Database Schema Configurator, you must have already prepared a database, user, table, and data in the database.

Each database connection schema set is shown as a single document

For example, the IDE attempts to gain the file lock when opening the schema set for a selected database connection. Each database connection schema set is shown as a single document

If any other user has already acquired the lock, then a **Lock Conflict** dialog box opens.

- Click **Yes** to acquire the lock.
- Click **No** to open the connection as read-only. Only users who acquire the write lock can import, synchronize, and save the schemas.

BLOB/CLOB support

The large object BLOB/CLOB is supported on the Database Schema Configurator. **BLOB** and **CLOB** are listed on the **Schema Data Type** list.

Oracle, MS SQL SERVER, and SQLite support the BLOB/CLOB datatype in the Cloverleaf Database protocol. The other database BLOB/CLOB datatype operations in Cloverleaf could vary, subject to its implementation against the JDBC standard.

For additional information, see [BLOB and CLOB](#) on page 366.

VRL file for database schema

When you import table/view schema, all visible columns are saved into a VRL file.

Database schemas are imported into Cloverleaf and stored in VRL format. To parse data from a file, the fields and line terminations must be recognizable to the corresponding VRL format. Fields must be separated with "," and rows must be separated with "\x0d\x0a."

XML schema file for table/view definition

When you import a table/view schema, the table/view schema is saved into a XML file. This file is prepackaged by the installation under `$HCIR00T\formats\dbschema`.

The `dbschema.xsd` file table schema file is located at `$HCISITE\formats\dbschema\sqlserver_connection`.

Folder structure and the ini file

Each database connection has an individual folder in which to store the table/view schemas.

The `dbconfiguration.ini` file is used to store all the database connection configurations. This is under the `$HCISITE` folder.

Each section is a database connection configuration.

The password key is 64-bit encoded.

BLOB and CLOB

In the database table, the BLOB type supports the binary data fetch or insert operations. The CLOB type supports variable-length large character object fetch or insert operations.

The BLOB/CLOB are supported by the engine message content and its cache file.

When the Cloverleaf database protocol processes the BLOB/CLOB objects, the engine caches the BLOB/CLOB data to the local file. It then saves that file path in the engine message to improve the message flow performance.

The message's BLOB/CLOB field is built by:

- The Cloverleaf CLOB or BLOB digest as the message content prefix.
- The full path of the cache file.
- (Optional) The timestamp for the GUID identifier attached to the digest.

The Cloverleaf BLOB digest is:

```
924C674350DE16494C901601E1504E96627757E61684CDF28B9A3E2847DE687A
```

The Cloverleaf CLOB digest is:

```
3A45E466D51049A11BCC3E52AD22216DE2E9046C4E98EA7545B60B8617487DE2
```

For CLOB inserting, the matched VRL field must start with the Cloverleaf CLOB digest with a full file path. The file content is text-based and should be UTF-8 encoded. For BLOB inserting, the matched VRL field must start with the Cloverleaf BLOB digest with a full file path. Usually, this is a binary file. The attached file is then loaded and inserted into the table's CLOB/BLOB column at engine runtime.

For example, an edited raw message is similar to:

```
1,3A45E466D51049A11BCC3E52AD22216DE2E9046C4E98EA7545B60B8617487DE2/opt/cis20.1/integrator/hel
loworld/LOBs/msgbig.txt,
924C674350DE16494C901601E1504E96627757E61684CDF28B9A3E2847DE687A/opt/cis20.1/integrator/hel
loworld/LOBs/releasenotes.pdf
```

msgbig.txt is inserted into the database table's CLOB column.

releasenotes.pdf is inserted into the database table's BLOB column.

The table schema is required in the Inbound database protocol when inserting/updating the CLOB/BLOB message into the database using the Read Success/Failure action.

Note: The CLOB/BLOB full file path is deleted after it is successfully inserted into the database at engine runtime. This is not deleted in `hcidbprotocoltest`, even though the “-r” option is used.

For CLOB fetching, the column's content is saved into a temporary file located within `$HCISITEDIR/exec/processes/processName/dbpcache` folder. The temporary file name ends with “.clb”.

The BLOB is the same as CLOB, but ends with “.blb”.

Both file names are combined with these part:

- A constant beginning with "dbpcache".
- An intermediate with sha256sum of the file itself. The timestamp at the end of sha256sum is optional.
- The postfix is last.

For example, a qualified BLOB field in the message resembles:

```
924C674350DE16494C901601E1504E96627757E61684CDF28B9A3E2847DE687A/opt/cis20.1/integrator/hello
world/exec/
processes/helloworld/dbpcache/dbp
cache38ec80f0caa60bb6d0a2c2ac1e39f45ea361e1e4d4208d79fae50f39c31a4edc.blb
```

Updating a database configuration

- 1 Select **Site Preferences > Database Configurations** to add a database configuration connection.
- 2 Click the Database Schema Configurator on the Launch Bar to open the **Select Database Connection** dialog box.
- 3 Select a configuration from the **Database Connection** list and click **OK** to open the Database Schema Configurator.

The Table Schema List panel contains all currently imported schemas under the database connection, sorted by schema name. Multiple selections are supported.

The Column List panel shows all columns of the currently selected schema. Selected columns are visible to the **Network Configurator**, **HRL Configurator**, and **Translation Configurator** dialog boxes.

Selecting a column's check box indicates that the column is added to the .vrl file for current table schema. By default, all columns are selected. Clicking **Name** clears all check boxes.

Each column has name, database data type, and schema data type. Sort the Column List table by clicking on a table header.

Required columns are selected and not available because they are always visible. For the schema data type attribute, possible values are those data types of VRL Format.

The Database protocol is based on the VRL format to implement. All selected columns in the Database Schema Configurator are converted to a VRL field. Column names must follow the naming rule of a VRL field. Only letters, digits, and underscores are supported. Cloverleaf does not support special characters, including spaces for table/column names, even though they are supported by the SQL standard syntax.

- 4 Use the editing buttons at the bottom of the Table Schema List as necessary.
 - **Import** adds schemas into the Table Schema List.
See [Importing table/view schema](#).
 - **Synchronize** synchronizes the selected table schemas with the database.
See [Synchronizing the selected table schema](#).
You can also use **File > Synchronize All**.
 - For **Add SQL Statement Schema/ Update SQL Statement Schema**, see [Adding/Updating SQL statement schema](#).

Column and table names

In the Database Schema Configurator, invalid column names are disabled in the column list table. The tooltip is a reminder of these disabled column names to ensure you clear the state.

The tooltip is similar to: "Cloverleaf does not support special characters, including spaces, for table/column names even though they are supported by the SQL standard syntax."

In table and column names, CIS only supports letters, digits, and underscores.

Importing table/view schema

- 1 Click **Import** to add the schemas into the Table Schema List. This is always enabled.
If the current database connection is opened as read-only, then a warning message pops up.
Otherwise, the **Import Table Schema** dialog box opens.
- 2 Select a table or view to populate Column List.
This shows all the tables and views under the current database connection, and all the columns of the currently selected table/view. Tables are sorted by clicking the column header.
- 3 Click **Refresh** to reload the table/view list.
- 4 Select a table or view.
- 5 Click **OK** to add the selected table/view schemas into the Table Schema List of the Database Schema Configurator.

Note: **OK** is not available until a table or view is selected.

- By default, all columns of the imported tables/views are visible and the corresponding Schema Data Type is String.
- The valid table/view name only accepts letter, digits, and underscores. Invalid table/views are disabled and cannot be selected from the left pane.
- If the selected table/view is already on the Schema List, then a message pops up to ask for confirmation to overwrite the existing entry.

Note: When using the **Table Schema Properties** dialog box (**Database Schema Configurator > Options > Table Schema Options**), double quotes are not supported for **Field Separator** or **Termination** for table schema.

Synchronizing the selected table schema

- 1 Click **Synchronize** under the Table Schema List to open the **Synchronize Table Schema** dialog box, which synchronizes the selected table schemas with the database. This is only enabled when a table schema is selected.
If the current connection was opened as read-only, then a message dialog box opens to remind you.
Otherwise, a **Synchronize Table Schema** dialog box opens.
- 2 Select how to synchronize.

By default, **Delete nonexistent columns** and **Automatically add new columns** are cleared.

If **Delete nonexistent columns** is selected, then the IDE compares each column and removes the ones in the schema that no longer exist in the connected database.

If **Automatically add new columns** is selected, then new columns are added into the table schema. By default, the added column is set as visible and the corresponding schema data type is set to String.

- 3 Click **OK** to begin the synchronization according to the settings.

For the Database Data Type and Required column attributes, the IDE compares each column. Then, it updates the ones in the table schema according to the ones in the connected database.

If the selected schemas are nonexistent in the source database, then they are not updated and a message dialog box pops up after synchronization.

If any schema are changed, then the current Database Schema Configurator is marked as modified.

After the synchronization starts, a progress bar is shown in the dialog box status bar to indicate the progress of the synchronization.

Synchronizing all table schemas

- 1 Click **Synchronize All** on the toolbar to open the **Synchronize Table Schema** dialog box. This synchronizes all the table schemas under the current database connection and follows the same steps as table schema synchronization.

Note: **File > Synchronize All** is enabled only when the Table Schema List contains values.

- 2 Select how to synchronize.

The **Synchronize Schema Configuration** dialog box has an added **Delete nonexistent table schemas** check box. When selected, this deletes from the Table Schema List all table schemas that no longer exist on the database. By default, the check box is not selected.

- 3 Click **OK** to begin the synchronization according to the settings.

Generating database schema from SQL statements

When you configure a complex inbound action SQL statement to compose a message, the imported table schema must match.

A specified schema is the schema that is generated from a SQL statement and has a customized name. This is generated from the SQL statement for the individual database thread.

Note: When generating SQL statement schema in the **Database Schema Configurator**, the SQLite Database Data Type column does not show any data. This does not happen with Oracle and SQLServer. This only happens with SQLite and does not effect any other operation that is related to SQLite SQL statement schema.

In database protocol thread configuration, you can compose a message with specific columns. To do this, define a SQL Statement Schema to only include those specific columns and apply it to the database protocol thread.

For example, a table named Table1 is in the database, and contains column1, column2, and column3.

To use only column1 and column3 to compose the database inbound message:

- 1 Click **Add SQL Statement Schema** to open the **SQL Statement Schema Configurator** dialog box.
- 2 Specify a name for the SQL statement schema in **Name**.
- 3 For **SQL Statement**, specify `select column1, column3 from table1` and click **OK**. The schema files are generated according to the SQL statement.
- 4 In Network Configurator, open the **Database Inbound Protocol Properties** dialog box and select the new SQL Statement schema from **Table Schema**. Other configuration steps are the same as that of table schema.

Adding/Updating SQL statement schema

These tools assist you in creating and modifying specified schema from a SQL statement for an individual database thread.

- 1 Select **Add SQL Statement Schema** to open the **SQL Statement Schema Configurator** dialog box.
- 2 Select **Simple SQL** or **Complex SQL** to open the **Inbound (Outbound) SQL Statement Configurator** dialog box to aid in defining SQL statements.
- 3 Make your selections and click **OK**.
This updates the **SQL Statement** field in the **SQL Statement Schema Configurator** dialog box.
- 4 Click **OK** on the **SQL Statement Schema Configurator** dialog box.
This adds the schema of the SQL statement into the Table Schema List on the main dialog box.
- 5 Select **Update SQL Statement Schema** to open the **SQL Statement Schema Configurator** dialog box.
This is filled by the selected SQL statement schema entry from the Table Schema List. This is where you can update the SQL statement or entry name.
- 6 Click **OK** to regenerate and update the SQL statement schema.

Editing table schema

On the Column List, select/clear a column to indicate which columns are to be visible.

For the Schema Data Type column, you can change the value by clicking in the cell and selecting from the menu.

Copy and paste operations in the Table Schema List can cross sites and IDE instances. You can copy a schema and paste it into another site through the same or another IDE instance.

Saving configurations

When saving all added/modified/deleted schemas on the Table Schema List, if the current connection is opened as read-only, a warning message opens. Otherwise, it attempts to gain the connection lock for the current user.

If the user does not own the lock on the current connection, then a **Lock Conflict** dialog box opens. Only a user who owns the lock can save the schemas. In all other cases a warning message opens.

If the current connection is used to save the database schemas for the first time, then a sub-folder with the connection name is created under `$HCISITE\formats\dbschema`.

When saving table schema, added and modified table/view schemas are saved into individual `schema_name.xml` files.

All visible columns are saved into `schema_name.vr1`. If a schema has no visible columns, then a message dialog box opens asking for confirmation to continue. When you select **Yes**, for table schemas without visible columns the corresponding `.vr1` file is removed and the `.xml` file is updated.

All of these files are stored under `$HCISITE\formats\dbschema\db_connection_name`.

Table schema options

The **Table Schema Properties** dialog box is for specifying a special field separator or termination character in your database schema format definition.

Multiple characters are permitted for termination of a database schema format. The newline default in Windows is CRLF, and in UNIX is LF.

Note: When using the **Table Schema Properties** dialog box (**Database Schema Configurator > Options > Table Schema Options**), double quotes are not supported for **Field Separator** or **Termination** for table schema.

Inserting and updating DATE/DATETIME database data

Data type columns are DATE or DATETIME. If one of these is in a database table, then you must set the appropriate **Schema Data Type** in the **Database Schema Configurator** dialog box. After this, the Database Outbound protocol can correctly insert/update data on a date type column.

For example, database tableA has a DATE type column `columnA` and the incoming message format for this column is MM/DD/YY. This means you must select **MM/DD/YY** as the **Schema Data Type**.

Note: For SQLite databases, you must use the SQLite function `datetime()` in the SQL statement/stored procedure to insert/update a DATETIME column.

- 1 For a SQLite database, if a table column's Database Data Type is DATETIME, select **String** as the Schema Data Type. To do this, click in the Schema Data Type column to open a list of choices.
- 2 Use the SQLite function `datetime('YYYY-MM-DD HH:MM:SS.SSS','localtime')` in the SQL statement to update the date type column.
- 3 Insert this into `tableA(ID, columnA) values(<tableA.ID>, datetime(tableA.columnA,'localtime'))`, where `tableA.columnA` is a String Schema Data Type with format YYYY-MM-DD HH:MM:SS.SSS.

Examples of using database schema in the Database Inbound protocol

In the Database Inbound protocol, database schema is used as the Cloverleaf message format. The Database Inbound protocol driver converts the result set that is returned from the SQL statement/stored procedure into a Cloverleaf message. This is based on the database schema.

These examples show how the Database Inbound protocol generates a Cloverleaf message based on database schema. The `Table_sanity` database schema includes visible and invisible columns.

Visible columns include ID, strFLD, intFLD, dateFLD, and flag.

The invisible column is testFLD.

Example 1: The returned result set includes all visible columns of the `Table_sanity` database schema

SQL statement:

```
select dateFLD, flag, ID, intFLD, strFLD from Table_sanity where Table_sanity.flag=0 order by Table_sanity.ID
```

Generated Cloverleaf message:

```
key01,test1,1,10/01/12,1
```

The field order of the generated Cloverleaf message is based on database schema's column order; it is not based on the result set metadata's column order.

If you set an empty database schema, then the Database Inbound protocol driver generates Cloverleaf messages based directly on the returned result set metadata.

Example 2: The returned result set includes part of the visible columns of the `Table_sanity` database schema

SQL statement:

```
select dateFLD, ID from Table_sanity where Table_sanity.flag=0 order by Table_sanity.ID
```

Generated Cloverleaf message:

```
key01, , ,10/01/12,
```

If the visible column that is defined in database schema cannot be retrieved from the result set, then the field value is set to empty. This is in the generated Cloverleaf message.

Example 3: The returned result set includes invisible columns that are not defined in the Table_sanity database schema

SQL statement:

```
select * from Table_sanity where Table_sanity.flag=0 order by Table_sanity.ID
```

Generated Cloverleaf message:

```
key01,test1,1,10/01/12,1
```

The Database Inbound protocol driver only retrieves visible columns that are defined in database schema from the result set. Other columns in the result set are ignored by the protocol driver.

Examples of using database schema in the Database Outbound protocol

In the Database Outbound protocol, database schema is used to parse the Cloverleaf message into column values. Then, the column values can be used to replace the placeholders set in the SQL statement/stored procedure.

These examples show how the Database Outbound protocol passes a Cloverleaf message that is based on database schema.

The Table_sanity database schema includes visible columns for ID, strFLD, intFLD, dateFLD, and flag.

Example 1: The outbound message field count is the same as the visible column count of database schema

Outbound message:

```
key01,test1,1,10/01/12,1
```

Database schema columns:

```
ID, strFLD, intFLD, dateFLD, flag
```

Parsing results

```
ID: key01
strFLD: test1
intFLD: 1
dateFLD: 10/01/12
flag: 1
```

If placeholder `tableName.columnName` is set in the SQL statement/stored procedure, the parsed column value replaces the corresponding placeholder in the SQL statement/stored procedure.

Example 2: The outbound message field count is less than the visible column count of database schema

Outbound message:

```
key01,test1, ,10/01/12
```

Database schema columns:

```
ID, strFLD, intFLD, dateFLD, flag
```

Parsing results:

```
ID: key01
strFLD: test1
intFLD:
dateFLD: 10/01/12
flag:
```

There are only four fields in the outbound message. No message field matches the flag database schema column, so the flag column has an empty value. The third field of the outbound message is empty, so the intFLD column also has an empty value.

Example 3: The outbound message field count is greater than the visible column count of database schema

Outbound message:

```
key01,test1,1,10/01/12,1,fakevalue
```

Database schema columns:

```
ID, strFLD, intFLD, dateFLD, flag
```

Parsing results:

```
ID: key01
strFLD: test1
intFLD: 1
dateFLD: 10/01/12
flag: 1
```

If the outbound message field count is greater than the column count of database schema, the extra fields in the outbound message are ignored.

Using SQL statements

This SQL is an example that shows how to use `to_char()` with the correct format. The SQL statement must be the same for steps 1 and 4.

This example is valid when the database that is used is Oracle, since `to_char` is a format function of Oracle.

- 1 In the Database Schema Configurator, click **Add SQL Statement Schema** to open the **SQL Statement Schema Configurator** dialog box.

- a For **Name**, specify **testFormat**.
- b For **SQL Statement**, specify this SQL statement:

```
SELECT to_char(CASE.ADMITDATE, 'YYYY-MM-DD HH24:MI:SS') as FORMAT_ADMINDATE, CASE.CASE_ID,
PATIENT.NAME, PATIENT.PATIENT_ID
FROM CASE, PATIENT
WHERE CASE.PATIENT_ID = PATIENT.PATIENT_ID
```

- 2 Click **OK** to close the dialog box.
- 3 Save the configuration.
- 4 In Network Configurator, select the database-inbound protocol for the thread to configure, and click **Properties**.
 - a In the **Database Inbound Protocol Properties** dialog box, for Table Schema select the **testFormat** table schema created in step 1.
 - b In the Read Action tab, select the **SQL Statement** option and specify the content listed in step 1.
- 5 Click **OK** to exit the dialog box.
- 6 Save the configuration.

Engine Output configurator

The Engine Output Configurator controls the type and amount of output information that is generated by the engine as it runs threads and processes. This information is reported by the engine in a log file.

There are two main steps to configuring engine output:

- Use Engine Output Configurator to create aliases that define the type and level of engine output.
- Use Network Configurator to apply those aliases to specific processes and threads.

EO (Engine Output) aliases control only informational and debug engine output. Even if no engine output aliases are configured, warnings and error messages are still written to the log file.

There are four types of information for EO aliases:

- Error
- Warning
- Information
- Debug

Error and Warning happen only in the log when there are errors or when warnings have happened.

For Information and Debug types, the system selectively prints messages that are based on the configuration, for example, `enable_info_all`, `enable_info_0`, `enable_info_1`, `enable_info_2`, and `enable_info_3`. The detail level goes from "0" to "3."

EO alias configuration

EO aliases define the type and level of output that is generated by specific threads and processes. The Engine Output Configurator lists all entries in the current alias definition.

For example:

```
ENABLE tps msgdisp INFO 2
```

- `ENABLE` is the action on the output.
- `tps` is the module affected by the entry.
- `msgdisp` is the submodule affected by the entry.
- `INFO` is the output severity.
- `2` is the level of detail for the output.

ENABLEALL_EXCEPT_MSG

This EO alias is used when you know there is an error, but are unsure what it is and require as much detail as possible.

Too much information, though, can fill up the log and could also contain patient data that should not be accessible by others.

In this case, you can use the `ENABLEALL_EXCEPT_MSG` EO alias

This alias includes everything in `enable_all`, except the message data.

Module and submodule selection

You can use the **Engine Output Entry** dialog box to select the modules and submodules to include in a configuration. Access this dialog box by clicking the **Append**, **Edit**, or double-clicking an entry on the Alias list.

Dialog box options	Description
Action	Click to select the output action for the current entry. Enable enables reporting of output from the specified modules and submodules. Disable disables output from the specified modules and submodules.
Module	Click Edit to open the Module Editor. Use this dialog box to select from a list of modules whose output is controlled by the current alias entry. An asterisk (*) specifies all modules.

Dialog box options	Description
Submodule	Click Edit to open the Submodule Editor . Use this dialog box to select from a list of submodules whose output is controlled by the current alias entry. An asterisk (*) specifies all submodules. Make selections as with the Module Editor.
Severity	<p>Select the appropriate check box to specify the severity or type of engine output.</p> <p>Info generates only user-oriented engine output information. All configured EO aliases should be at "Info."</p> <p>Debug generates only debug information. Do not select this option unless instructed to do so by Support.</p> <p>Selecting Info and Debug generates both types of output. Warnings and error messages are displayed, regardless of the EO alias configuration.</p>
Level	Click the arrow to open a list of detail choices. "0" gives the least detail, "3" gives the most detail, and "*" gives all details.
Save New	<p>This is available when adding a new entry to the alias list. Create a new entry and click Save New to accept the new entry. The Engine Output Entry dialog box remains open for the selection of additional entries. When all entries are added, click Save. The Engine Output Entry dialog box closes and the Engine Output Configurator shows the new entries.</p> <p>When accessing an existing entry through the Edit tool, this button is not available.</p>

Log level details

Log level	Description
0	<p>Conveys high-level information about the general state of what is happening in the engine. Log messages at this level tend to be generic and lack specific values.</p> <p>Examples:</p> <ul style="list-style-type: none"> • Initializing Protocol Driver • Purging GRM object cache • Placing msg on Pre-xlate queue

Log level	Description
1	<p>Conveys information that is still at a high level, but more specific than level 0. Often contains specific values such as thread names, message metadata values, and so on.</p> <p>Examples:</p> <ul style="list-style-type: none"> Processing msg using trxId '_HCI_stat-ic_route_' xlatePlaceOnPartialQue: msgDestThread = "conn_1"
2	<p>Provides a level of detail that gives insight regarding the logic that is happening in the engine.</p> <p>Example:</p> <p>SMAT "conn_1_ib" already exists, checking format for consistency Reading first line of index file: Unable to read because the file is locked. No timer event returned. Go back to scheduler.</p>
3	<p>Provides detailed information about the logical state of the engine or contains specific values for a particular message.</p>

Module editor

Click **Edit** next to the module and submodule text box on the **Engine Output Entry** dialog box. This opens the Module Editor. The Submodule Editor works the same as the Module Editor.

Use the selection box to select from a list of modules/submodules whose output is controlled by the current alias entry.

After a module has been selected, submodules can then be selected or cleared.

Modules/Submodules

This is a list of the modules/sub-modules and what they return in value when used with the Engine Output Configurator.

Note: The sub-modules vary depending on the selected module and are a function of the module selected.

Module	Submodules
appc Advanced Peer-to-Peer Communication; used for the LU6.2 protocol driver.	<ul style="list-style-type: none"> <code>close</code>: Close a file or connection <code>control</code>: Control message <code>init</code>: Initialization <code>open</code>: Open a file or make a connection <code>read</code>: Read data <code>write</code>: Write data
cmd Command; handles commands sent by <code>hcicmd</code> , that is, <code>pstart</code> , <code>pstop</code> , and <code>die</code> .	cmd : Command
dbi Database Interface; all functions relating to database operations, not ODBC.	<ul style="list-style-type: none"> <code>dbi</code>: Database Interface <code>elog</code>: Error database operation information <code>etps</code>: Error database TPS error <code>icl</code>: Inter-thread communication library <code>log</code>: Generic database operations, specifically saving message content to and from the database <code>rlog</code>: Recovery database operation information
dbp	<ul style="list-style-type: none"> <code>init</code>: Initialization <code>open</code>: Open a file or make a connection <code>read</code>: Read data <code>write</code>: Write data <code>close</code>: Close a file or connection
dicm	<ul style="list-style-type: none"> <code>init</code>: Initialization <code>open</code>: Open a file or make a connection <code>read</code>: Read data <code>write</code>: Write data <code>close</code>: Close a file or connection
diagModule Diagnosis; used for internal engine debugging	<ul style="list-style-type: none"> <code>diag</code>: Other diagnostic message <code>leak</code>: Memory allocation/deallocation; useful for diagnosis of memory leaks
dnet Decnet protocol drivers	<ul style="list-style-type: none"> <code>close</code>: Close a file or connection <code>init</code>: Initialization <code>open</code>: Open a file or make a connection <code>read</code>: Read data <code>write</code>: Write data

Module	Submodules
file File protocol drivers	<ul style="list-style-type: none"> close: Close a file or connection init: Initialization open: Open a file or make a connection read: Read data write: Write data
fset	<ul style="list-style-type: none"> close: Close a file or connection control: Control message init: Initialization open: Open a file or make a connection read: Read data write: Write data
http HTTP protocol driver	<ul style="list-style-type: none"> init: Initialization open: Open a file or make a connection read: Read data write: Write data tps: Tcl Procedure Stream
icl Interthread Communications Library; provides interthread communication (inter- or intra-process)	<ul style="list-style-type: none"> cb: Callback conn: Connection db: Database icl: Inter-thread communication library server: ICL has a server/client relationship between threads. This is for the server side information. tcpip: Transmission Control Protocol/Internet Protocol
java	<ul style="list-style-type: none"> close: Close a file or connection init: Initialization open: Open a file or make a connection read: Read data write: Write data
lu3 Logical Unit 3; IBM SNA LU3 protocol, used when Cloverleaf acts as a printer	<ul style="list-style-type: none"> close: Close a file or connection init: Initialization open: Open a file or make a connection read: Read data write: Write data

Module	Submodules
mqz IBM WebSphere MQ	<ul style="list-style-type: none"> • <code>close</code>: Close a file or connection • <code>init</code>: Initialization • <code>open</code>: Open a file or make a connection • <code>query</code>: Message data query • <code>read</code>: Read data • <code>write</code>: Write data
msg Message; routines related to management of message objects	<ul style="list-style-type: none"> • <code>mid</code>: Message ID • <code>msg</code>: Message object • <code>recdefload</code>: Message format definition loading • <code>recdegquery</code>: Data query on message • <code>recformat</code>: Data conversion • <code>recparse</code>: Message parsing • <code>tbl</code>: Lookup tables
msi Monitor Statistics Interface; routines related to monitoring and statistics	<code>msi</code> : Engine Monitor Statistics Interface
nci NetConfig interface	<code>nci</code> : Information for NetConfig-related issues
numf Numbered File protocol drivers	<ul style="list-style-type: none"> • <code>close</code>: Close a file or connection • <code>init</code>: Initialization • <code>open</code>: Open a file or make a connection • <code>read</code>: Read data • <code>write</code>: Write data
pd	<ul style="list-style-type: none"> • <code>close</code>: Close a file or connection • <code>control</code>: Control message • <code>general</code>: General • <code>init</code>: Initialization • <code>open</code>: Open a file or make a connection • <code>read</code>: Read data • <code>write</code>: Write data • <code>thread</code>: Internal thread
pdl Programmable Driver Language; routines used to control user-written PDLs	<ul style="list-style-type: none"> • <code>close</code>: Close a file or connection • <code>init</code>: Initialization • <code>open</code>: Open a file or make a connection • <code>pdl</code>: Programmable Driver Language • <code>read</code>: Read data • <code>write</code>: Write data
prod Product	<code>prod</code> : System environment; general configuration

Module	Submodules
pti Pthreads interface	<ul style="list-style-type: none"> event: Event msg: Message pti sched: Schedule select signal thread: Internal thread
sms Serial Module Stack (TPS Tcl Procedure Stream) This controls running of the user-defined TPS module.	sms: TPS stack processing information
tcp TCP/IP; length-encoded TCP/IP protocol drivers	<ul style="list-style-type: none"> close: Close a file or connection init: Initialization open: Open a file or make a connection read: Read data write: Write data
tii Tcl Interpreter Interface	tii: Information for Tcl-related issues
tps Tcl Procedure Stream	<ul style="list-style-type: none"> eodata: Engine Output data msgdisp: Message disposition tps: Tcl Procedure Stream
upoc User Point of Control; UPoC protocol drivers	<ul style="list-style-type: none"> init: Initialization open: Open a file or make a connection read: Read data write: Write data tps: Tcl Procedure Stream
util General utility functions	<ul style="list-style-type: none"> counter: Sequence counters fs: File system nd: Named data object utils: Utilities
xlt Translation thread	<ul style="list-style-type: none"> format: Data format route: Routing thread: Related to threading issues xlate: All other miscellaneous translation issues
xpm Translation (Xlation) Pseudo Machine	<ul style="list-style-type: none"> xlt: Translation xpm: Translation Pseudo Machine

Module	Submodules
xslt Extensible Stylesheet Language Transformations (XSLT)	xslt: XSLT processing

Selecting modules/submodules

- 1 Use the left and right arrows to place selected modules/submodules in the left column and unnecessary modules/submodules in the right column.
- 2 Click **Apply** when finished making selections. The **Engine Output Entry** dialog box displays showing the selected modules/submodules.
- 3 Configure as necessary in the **Engine Output Entry** dialog box.
For new entries, click **Save New** in the **Engine Output Entry** dialog box if there are additional entries to make. Otherwise, click **Save**. The Engine Output Configurator shows the new configuration.
For existing entries accessed through the **Edit** tool, click **Save**. The Engine Output Configurator shows the new configuration.

Editing an existing entry

- 1 On the Engine Output Configurator, double-click the entry to edit.
This opens the **Engine Output Entry** dialog box.
- 2 Edit as necessary.
- 3 Click **Save** in the **Engine Output Entry** dialog box.
The **Engine Output Entry** dialog box closes and the Engine Output Configurator shows the edited entry.
- 4 Save the file.

Adding a new entry

- 1 Select **Append**, **Add Before**, or **Add After**.
This opens the **Engine Output Entry** dialog box.
- 2 Select the modules and submodules to include in the new configuration.
- 3 Click **Save New** in the **Engine Output Entry** dialog box if there are additional entries to make. Otherwise, click **Save**. The Engine Output Configurator shows the new configuration.
- 4 Save the file.

Applying aliases with the Network Configurator

After creating an EO alias, you can use the Network Configurator to apply the alias to specific processes or threads. The EO aliases then control the type and amount of output that is generated by those processes and threads.

Applying an EO alias to a process

The threads, or connections, in the network operate within the environment of a process.

To apply an EO alias to a process, select **Process > Configure** in Network Configurator. This opens the **Process Configuration** dialog box. This is where you can assign a default alias to a process.

For **Default Engine Log Configuration**, click **List** to select from a list of aliases. Specify zero or more default EO aliases to apply to this process. These default EO aliases apply to all threads in the process, including the command thread, the translation thread, and all protocol threads.

For **Xlate Engine Log Configuration**, click **List** to select from a list of aliases. Specify zero or more EO aliases to apply to the translation thread of this process. There is only one translation thread per process.

The process default and xlate EO aliases are evaluated and applied in the listed order, from left to right. If any aliases conflict with one another, then the later entry overrides the earlier one.

Applying an EO alias to a thread

On the Network Configurator's **Properties** tab, in the Thread panel click **List** for **Engine Log Configuration**. In the **Engine Log Configuration** dialog box, select one or more EO aliases to apply to this thread.

The aliases are evaluated and applied in the listed order, from left to right. If any aliases conflict with one another, then the later entry overrides the earlier one.

The process default EO aliases are applied before the individual thread EO aliases. Conflicts between the aliases are resolved in favor of the last alias applied.

Applying an EO alias to an inbound message

On the Network Configurator **Properties** tab, in the Inbound pane click **List** for **Msg Engine Log Configuration**. In the **In Message Engine Log Configuration** dialog box, select one or more EO aliases.

This is used for all inbound data and reply messages for this connection. You can view only the lines of output that contain a message ID number.

The process default, translation, and any thread EO aliases are applied before the inbound message EO aliases. Conflicts between the aliases are resolved in favor of the last alias applied.

Viewing engine output

To view the engine output of a thread, select the **Watch Output** or **Browse Output** options from the **Thread Controls** dialog box of the Network Monitor.

Viewing the output of a process is similar to the thread, except that you right-click a process and select **Control > Full**.

Viewing the engine output of a thread:

- 1 In the View List pane of the Network Monitor, right-click the thread to view to open the context menu.
- 2 Select **Control** and then **Full**. This opens the **Thread Controls** dialog box for the selected thread.
- 3 Click **Watch Output** or **Browse Output**.
Watch Output opens a Watch Log File for the selected thread. EO aliases control the type and amount of output.
Browse Output opens a Log File for the selected thread. EO aliases control the type and amount of output. Use the **Search** text box at the bottom of the dialog box to look for specific text.

FRL Configurator

The FRL (Fixed-length Record Layout) Configurator defines how data is formatted within fixed-length record layouts.

An FRL definition represents a single record where all the fields are a fixed width. In the system, all FRL fields have at least one subfield. Subfields can be of variable length as long as they fit within a fixed length field.

FRL files reside in `$HCISITEDIR/formats/`.

Before configuring, specific transaction information is required, including:

- The types of transactions that are processed by each connection.
- The record layouts for those transactions.
- The types of data that are handled by each transaction.
- The way that data is processed or translated.

Features

With the FRL Configurator, you can:

- Build a list of fields and their attributes.
- Define the fixed and variable length subfields.
- Edit previously built fixed length record layouts.
- Define the validation rules.
- Define the masks that define the layout of commonly used record fields.
- Define the groups that define a group of related fields.

GUI features include:

- Location indicator
The Location pane shows the location of each field definition within the record. The indicator is a rectangle with two numbers and marks the start and end positions of the group, mask, field, or subfield.

- For a group, mask, or field, the Location Indicator is unfilled. This marks the start and end positions of the current group, mask, or field.
- For a subfield, the Location Indicator is dark gray. This marks the start and end positions of the subfield within the entire record.
- Variable-length (width) subfields
There are two instances where a prefix is used in a subfield:
 - Variable-width subfields. For example, "Doe, John."
 - Prefixed fields. For example, "mrn:1234567890."
- Parsing
Variable-width subfields are parsed using prefixes instead of delimiters. Although subfields are variable in width, they must have a defined maximum width, or length. Prefix characters are not considered part of the subfield's data.
- Subfield size
Specifications typically do not list subfield sizes for variable-width subfields.
For example, "Public, John Q":
 - The subfield last name has no prefix.
 - The subfield first name has a ", " (comma + space) prefix.
 - The subfield middle initial has a space prefix.

Useful menu bar options

Include > Add Include opens a dialog box from which you select a file to include in the record layout. The files for record layouts, translation specs, and lookup table, under the **Show** menu, reside in \$HCISITEDIR/formats.

For **Edit > Replicate**, when the focus is on a field or subfield, this places a copy of the highlighted field or subfield onto the Layout pane. Specify the **Count**, the number of replications, and **Starting Number** in the dialog box. The **Starting Number** is valid only for fields. Selecting **Replicate** when a subfield is highlighted opens a menu with only the **Count** selection.

Control > Auto-Recalc automatically recalculates the field width whenever changes are made to a subfield width. If this option is not selected, then the field width is not recalculated until **Recalculate Field Width** is clicked. By default, the FRL Configurator starts in auto-recalc mode.

Configuring the field definition

Fixed-length records consist of one or more fields, which can have one or more subfields. The fields and subfields can be right- or left-justified, and require fill or pad characters, such as spaces. Each field is defined by a unique name.

An FRL must match the data to or from the ancillary system, or incorrect results are produced.

- 1 To create a new file or open an existing one, select **Define > Define Fields**.
To create a new field, select **New Field** from the toolbar. This places a new field folder in the Layout pane.

To reconfigure an existing field, open the file to edit. In the Layout pane, click the field to reconfigure. Its current properties display in the Field Properties pane.

- 2 Specify the field name.
- 3 In **Field Char**, click the arrow to select the fill, or pad, character to use when creating or parsing the data in this field. The field justification determines which end of the data is padded with the fill character.
- 4 For **Width**, specify the total width of the field. If left blank, then field width is automatically calculated from the total width of the subfields when **Auto-Recalc** is enabled on the **Control** menu.
For special requirements, such as pad characters, specify a total field width that is greater or smaller than the sum of the subfield widths.
- 5 For **Recalculate Field Width**, click to recalculate the field width from its subfields. This is only required when the **Auto Recalc** feature is off.
- 6 For **Justification**, select **Left Justify** or **Right Justify** for the field. The parser strips all padding characters on the non-justified side. For example, a field is right-justified and "0" is the padding character. The field before parsing is "00123," and the field after parsing is "123."
- 7 Select the validation.
Validate at Fetch. Click to validate the field whenever this record layout is retrieved or used for an input or output operation. With this selected, messages with an invalid field are parsed, but a warning is given.
Validate at Parse. Click to validate the field when the data is parsed. This option forces validation even when the field is not used. With this selected, messages with an invalid field fail when parsed.
- 8 Select the field type:
 - **Normal** defines the field format by the current field entries, instead of a mask or group. This is the default setting.
 - **Mask** defines the current field with a mask. The menu lists available masks.
 - **Group** defines the current field with a definition (group). The menu lists available groups.

Configuring field subfield properties

- 1 To create a new subfield, select the field in which to place the subfield and click **New Subfield**.
Each new field contains one subfield as the default. Additional subfields are created by clicking **New Subfield** on the toolbar.
To reconfigure an existing subfield, open the field in the Layout, then select the subfield to reconfigure.
- 2 Open the **Data Type** menu to select a data type. Define all subfields as **string** unless date conversion is required during translation.
When a date data type is selected, the field length is automatically filled with a default value for that data type. Date subfields are automatically converted from one format to another during translation.
For **Julian**, the format depends on the width of the field:
 - A width of 5 represents YYJJJ.
 - A width of 7 represents CCYYJJJ.
 Most of the date formats depend on the width. Adding 2 characters to the width adds the century. This applies to formats that do not specify the century. For example, YYMMDD is 6 characters by default. If this is changed to 8 characters, then the format becomes CCYYMMDD.

Zoned Decimal preserves the decimal.

For **Numeric**, all characters must be numeric.

- 3 For **Fill Char**, open the menu to select the fill, or pad, character to use when creating or parsing the data in this field. The field justification determines which end of the data is padded with the fill character.
- 4 For **Prefix**, specify the characters that precede this subfield, to separate it from the previous subfield in the record layout. This attribute is used only for variable-width subfields.
- 5 For **Width**, specify the maximum width of the subfield, not including any prefix.
- 6 For **Justification**, Select **Left Justify** or **Right Justify** for the field. The parser strips all padding characters on the non-justified side. For example, a field is right-justified and "0" is the padding character. The field before parsing is "00123," and the field after parsing is "123."
- 7 For **Validation**, select how to validate the subfield data whenever this record layout is used as an input or output record.
 - No Selections. No validation.
 - **Existence**. Validates the existence of the data, but not its content.
 - **Contents**. Validates the type of data, but not its existence.
 - **Existence+Contents**. Validates both the existence and the content of the data.

Configuring a mask

A mask is a template that is used to define data fields that have the same layout. This contains a single field definition that has a set of subfields.

Create masks to define field formats used in multiple places. Each mask is composed of one or more subfields, which contain the actual data within the record.

Defining a mask is similar to defining a field within a record layout.

- 1 Create a new file or open an existing one, then select **Define > Define Masks**.
To create a new mask, select **New Field** from the toolbar. This places a new folder in the Layout pane.
To reconfigure an existing mask, open the file to edit. In the Layout pane, click the mask to reconfigure.
- 2 Specify the mask name.
- 3 For **Fill Char**, click the arrow to select the fill, or pad, character to use when creating or parsing the data in this field.
The field justification determines which end of the data is padded with the fill character.
- 4 For **Width**, specify the total width of the field.
If left blank, then field width is automatically calculated from the total width of the subfields when **Auto-Recalc** is enabled on the Control menu.
For special requirements, such as pad characters, specify a total field width that is greater or smaller than the sum of the subfield widths.
- 5 Click **Recalculate Field Width** to recalculate the field width from its subfields.
- 6 For **Justification**, select **Left Justify** or **Right Justify**. The parser strips all padding characters on the non-justified side.

For example, a field is right-justified and "0" is the padding character. The field before parsing is "00123" and the field after parsing is "123."

7 For **Validate** select from:

- **Validate at Fetch.** This validates the field whenever this record layout is retrieved or used for an input or output operation. With this selected, messages with an invalid field are parsed, but a warning is given.
- **Validate at Parse.** This validates the field when the data is parsed. This option forces validation even when the field is not used. With this selected, messages with an invalid field fail when parsed.

Configuring mask subfield properties

1 Select the mask folder in which to place the subfield and click **New Subfield**.

Each new mask contains one subfield as the default.

2 For **Data Type**, click the arrow to select from a menu of data types. Define all subfields as **String** unless date conversion is required during translation.

When a date data type is selected, the field length is automatically filled with a default value for that data type. Date subfields are automatically converted from one format to another during translation.

For **Julian**, the format depends on the width of the field:

- A width of 5 represents YYJJJ.
- A width of 7 represents CCYYJJJ.

Most of the date formats depend on the width. Adding 2 characters to the width adds the century, in formats that do not specify the century. For example, YYMMDD is 6 characters by default. If this is changed to 8 characters, then the format becomes CCYYMMDD.

Zoned Decimal preserves the decimal.

For **Numeric**, all characters must be numeric.

3 For **Fill Char**, click the arrow to select the fill, or pad, character to use when creating or parsing the data in this field. The field justification determines which end of the data is padded with the fill character.

4 For **Prefix**, specify the characters that precede this subfield. This separates it from the previous subfield in the record layout. This attribute is used only for variable-width subfields.

5 For **Width**, specify the maximum width of the subfield, not including any prefix.

6 For **Justification**, select **Left Justify** or **Right Justify** for the field. The parser strips all padding characters on the non-justified side. For example, a field is right-justified and "0" is the padding character. The field before parsing is "00123" and the field after parsing is "123."

7 For **Validation**, select how to validate the subfield data whenever this record layout is used as an input or output record.

- **No Selections.** No validation.
- **Existence.** Validates the existence of the data, but not its content.
- **Contents.** Validates the type of data, but not its existence.
- **Existence+Contents.** Validates both the existence and the content of the data.

FRL group definition

A group is a set of logically related data fields. It is used to define a multiple field format in multiple places, such as full addresses or insurance data.

After defining a group, use it as a template for declaring groups of data fields in the record layout.

Configuring group field properties

Both groups and masks serve as templates for defining the data fields in a record layout. A mask defines only a single data field. A group defines many related fields.

A group is a set of logically related data fields. It is used to define a multiple field format in multiple places, such as full addresses or insurance data.

After defining a group, you can use it as a template for declaring groups of data fields in the record layout.

- 1 Select **Define > Define Groups** and specify a group name.
- 2 Select **New Field** from the toolbar and specify the field name.
- 3 For **Fill Char**, click the arrow to select the fill, or pad, character to use when creating or parsing the data in this field.

The field justification determines which end of the data is padded with the fill character.

- 4 For **Width**, specify the total width of the field. If left blank, then field width is automatically calculated from the total width of the subfields when Auto-Recalc is enabled on the Control menu. For special requirements, such as pad characters, specify a total field width that is greater or smaller than the sum of the subfield widths.
- 5 For **Relative Start**, specify the number of spaces to add before the data in this field.
- 6 Click **Recalculate Field Width** to recalculate the field width from its subfields.
- 7 For **Justification**, select **Left Justify** or **Right Justify** justification for the field.

The parser strips all padding characters on the non-justified side. For example, a field is right-justified and "0" is the padding character. The field before parsing is "00123" and the field after parsing is "123."

- 8 Select the validation:
 - **Validate at Fetch**. Click to validate the field whenever this record layout is retrieved or used for an input or output operation. With this selected, messages with an invalid field are parsed, but a warning is given.
 - **Validate at Parse**. Click to validate the field when the data is parsed. This option forces validation even when the field is not used. With this selected, messages with an invalid field fail when parsed.
- 9 Select a field type:
 - Select **Normal** to define the field format by the current field entries, instead of a mask or group. This is the default setting.
 - Select **Mask** to define the current field with a mask. Click the arrow to open a menu listing available masks.

Configuring group subfield properties

Each field of the group is composed of one or more subfields. Subfields contain the actual data within the record. The subfields of a field are the same as the subfields of a record layout data field.

- 1 Select the field in which to place the new subfield. Then, select **New Subfield** from the toolbar.
Each new field contains one subfield as the default. Additional subfields are created by clicking **New Subfield**.
- 2 For **Data Type**, click the arrow to select from a menu of data types. Define all subfields as **String** unless date conversion is required during translation.
When a date data type is selected, the field length is automatically filled with a default value for that data type. Date subfields are automatically converted from one format to another during translation.
For **Julian**, the format depends on the width of the field:
 - A width of 5 represents YYJJJ.
 - A width of 7 represents CCYYJJJ.
 Most of the date formats depend on the width:
Adding 2 characters to the width adds the century, in formats that do not specify the century. For example, YYMMDD is 6 characters by default. If this is changed to 8 characters, then the format becomes CCYYMMDD.
Zoned Decimal preserves the decimal.
For **Numeric**, all characters must be numeric.
- 3 For **Fill Char**, click the arrow to select the fill, or pad, character to use when creating or parsing the data in this field. The field justification determines which end of the data is padded with the fill character.
- 4 For **Prefix**, specify the characters that precede this subfield. This separates it from the previous subfield in the record layout. This attribute is used only for variable-width subfields.
- 5 For **Width**, specify the maximum width of the subfield, not including any prefix.
- 6 For **Justification**, select **Left Justify** or **Right Justify**. The parser strips all padding characters on the non-justified side.
For example, a field is right-justified and "0" is the padding character. The field before parsing is "00123" and the field after parsing is "123."
- 7 For **Validation**, select how to validate the subfield data whenever this record layout is used as an input or output record.
 - No Selections. No validation.
 - **Existence** validates the existence of the data, but not its content.
 - **Contents** validates the type of data, but not its existence.
 - **Existence+Contents** validates both the existence and the content of the data.

FRL data types

This table lists the most commonly used fill characters for flat or fixed-length record layouts:

Data type	Usage/Description
Integer	Use this for doing math on numbers.
String	Use this for most purposes, unless you must do math.
Julian Date	This is a continuous count of days and fractions since noon Universal Time on January 1, 4713 BCE, on the Julian calendar.
MMDDYY	Y = Year; M = Month; D = Day
MM/DD/YY	Y = Year; M = Month; D = Day
YYDDMM	Y = Year; M = Month; D = Day
DDMMYY	Y = Year; M = Month; D = Day
DD.MM.YY	Y = Year; M = Month; D = Day
HHMMSS+Z	H = Hour; M = Minute; S = Second; Z = Offset from Coordinated Universal Time (UTC)
YYMMDDHMS+Z	Y = Year; M = Month; D = Day; H = Hour; M = Minute; S = Second; Z = Offset from Coordinated Universal Time (UTC)
Zoned Decimal	Specifies a method of encoding decimal numbers in which each digit requires one byte of storage. The last character contains the number's sign and the last digit.
Numeric	Data string

FRL fill characters

This table lists the most commonly used fill characters for flat or fixed-length record layouts.

Fill character	Description
\x00	NUL null
\x01	SOH start of heading
\x02	STX start of text

Fill character	Description
\x03	ETX end of text
\x04	EOT end of transmission, not ETB
\x05	ENQ enquiry
\x06	ACK acknowledge
\x07	BEL bell
\x08	BS backspace: move print head or cursor back one space
\x09	TAB horizontal tab: move print head or cursor to the next tab on the right
\x0A	LF line feed: move print head or cursor to a new line
\x0B	VT vertical tab: move print head or cursor down one line
\x0C	FF form feed: advance paper to the top of the next page
\x0D	CR carriage return: move print head or cursor to the leftmost column
\x0E	SO shift out: change to alternate character set
\x0F	SI shift in: revert to default character set
\x10	DLE data link escape
\x11	DC1 device control 1

Fill character	Description
\x12	DC2 device control 2
\x13	DC3 device control 3
\x14	DC4 device control 4
\x15	NAK negative acknowledge
\x16	SYN synchronous idle
\x17	ETB end of transmission block, not EOT
\x18	CAN cancel
\x19	EM end of medium
\x20	SP space
\x1A	SUB substitute
\x1B	ESC escape
\x1C	FS file separator
\x1D	GS group separator
\x1E	RS record separator
\x1F	US unit separator
\x7F	DEL delete

FRL field definition fill characters

Fill characters in FRL field definition are handled as SP-0x20 by the engine.

On the CIS GUI, when an FRL file is defined using the FRL Configurator, a value is not set for the **Fill Char** field. The fill characters in the .frl file are an empty value.

In the runtime, the engine sees the fill character as SP-0x20.

This is the engine behavior. Some sub-fields do not display when the sub-field value is only SP-0x20 and **Fill Char** is not set as SP-0x20.

FRL record layout definition example

Before configuring an FRL record layout, specific transaction information is required, including:

- The types of transactions that are processed by each connection
- The record layouts for those transactions
- The types of data that are handled by each transaction
- The way that data is processed or translated

Fixed-length records consist of one or more fields, which can have one or more subfields. The fields and subfields can be right or left justified, and require fill or pad characters, such as spaces. Each field is defined by a unique name.

An FRL must match the data to or from the ancillary system, or incorrect results are produced.

Use the FRL tab to test fixed-record layout configurations. This test reads, parses, and displays messages, ensuring that configurations match the input data.

Access `hcifrltest` directly from the command line or through the Testing Tool.

Creating a fixed-length flat record layout

Note: Fields and subfields must be in precise order in the Layout pane for the FRL layout to properly parse or encode data.

- 1 Click **New Field**. A new field displays in the Layout pane.
- 2 In Field Properties, specify the field name in **Name**.
- 3 Click **Apply**. The name of the selected field changes in the Layout pane.
- 4 Select the appropriate subfield beneath the field in the Layout pane.
- 5 Update the Subfield Properties as necessary and then click **Apply**.
- 6 Repeat steps 2-5 to create an FRL layout with the field and subfield properties that are listed in the data table.
- 7 Save the layout.

Testing the layout

- 1 Click **Test** in the FRL Configurator toolbar. The Testing Tool opens with the FRL tab active.
- 2 Select the FRL layout file from the **FRL File** menu.
- 3 In the Test Options area, click **Open**, the folder icon, next to **Choose Data File**.
- 4 Navigate to the data folder and select the data file with which to test. Click **Open**. The path to the data file displays in **Choose Data File**.
- 5 Select the data file with which to test. Click **Open**. The path to the data file displays in **Choose Data File**.
- 6 Select the **Detail Level** from the menu list.
- 7 Select **process one record**.
- 8 In the Preview Command to Issue area, click **Run Command**. The output displays in the Result area.

Global Variables Configurator

Global variables can help you simplify development and deployment of Cloverleaf interfaces.

Global variables are string name-value pairs that are used to define commonly-used or environment-related values for a site. After there are changes to variable values, these changes are then applied to all areas where the variable is used.

The engine runtime searches and replaces global variable names with actual values. Any string that matches the global variable naming pattern that fails to be replaced is reported as a warning. The string is left unchanged.

Global Variables support the master site model. For example, a global variable defined in a local site takes precedence over the same variable defined in the master site.

Global variables cannot be used recursively, that is, a global variable cannot be a part of the value of another global variable.

The Global Variables Configurator is similar to the Lookup Table Configurator. Global variable values can be masked and encrypted similar to lookup table entries.

Every variable must have a \$\$ prefix. For example, \$\$prodadtpport.

Global variables can be used in configuration file fields such as:

- NetConfig protocol parameters: For example, port number, directory path, and web services URL.
- Xlate files: Constants in input fields
- Lookup tables: Input or output values
- Arguments to UPoCs
- Alert definition entries

Global variable names are restricted to:

- A-Z
- a-z
- 0-9

- `_` (underscore)

This tool lists all variables in an editable table. You can manipulate table cells to edit any variable name and value. Then, you can set whether the value should be encrypted before it is saved into `globalVariables.ini`.

Table values are masked when encrypted is selected, unless the password is provided. A right-click menu is available to edit variables.

Variables can be deleted and variable names can be updated, whether the variable is being used. When doing this, there could be an issue if an active variable is removed. To remind users of potential issues, the IDE tools that support global variables have additional validation, and prompts for any missing global variables.

Global variables affect other functions in these ways:

- CLAPI has interfaces for getting and setting global variables.
- The Site Document has a page to display all global variables.
- For NetConfig, lookup tables, alerts, and xlate files where the global variables are used, the variable names are a link to the Global Variables tool. These links are located in Site Document.
- Security Server can add the tool and the `ini` file under security control.
- The `globalVariables.ini` is under Version Control, and has file read/write lock support.
- There are Tcl/Java APIs to get variable values (read-only).

When **Set Password** on the **File** menu is configured, you can view all encrypted values only when a correct password is provided. Password values and global variables marked as encrypted are encrypted when saving into `globalVariables.ini`.

Data flow

When the engine/monitorD/tool starts, the configuration file with defined global variables is loaded into the engine/monitorD/tool.

When there are other configuration files, it replaces the global variables in each configuration file loading a process.

With the NetConfig file, it replaces the global variable when it gets the value or replaces all variables before parsing it as a Tcl object.

For xlate files, it does the global variable replacement in the xlate file compiling period.

The configuration file loader in the engine/monitorD/tool replaces the global variable with the defined real value in the global variable definition file. Encrypted values in the global variable file are first decrypted and then used for replacement by the loader.

If the engine/monitorD/tool fails to do the replacement, then a warning is issued in the log.

Information storage

Global variables that are packaged into a BOX at the site level are saved to `HCIR00T/box/BoxDir/BoxName/AppDefaults/globalVariables.ini`.

For the mastersite level, global variables are saved to `HCIR00T/box/<BoxDir>/<BoxName>/_mastersite/AppDefaults/globalVariables.ini`.

There is no default `ini` file when a site is initialized; the file is created when you configure global variables for the first time and save through the IDE.

You can view all encrypted values only when the correct password is provided in the Global Variable Configurator.

Password values and global variables marked as encrypted are encrypted when saving into `globalVariables.ini`.

Encryption

For configuration files, environment-related encryption/decryption keys are used. This means encrypted configuration data in one cloverleaf environment cannot be decrypted in another environment.

BOX creation and deployment performs these steps, that encrypted values in `globalVariables.ini`, or other files that have encrypted data, can be ported to other environments.

When creating a BOX, the system:

- 1 Gets a copy of `globalVariables.ini` which contains all referenced variables.
- 2 Decrypts the sensitive data with environment-related keys to get plain text values.
- 3 Encrypts the plain text values with the public key, which is not environment-related.
- 4 Saves the encrypted values into the `ini` file to BOX.

When deploying a BOX, the process is reversed, where the system:

- 1 Reads `globalVariables.ini` from BOX.
- 2 Decrypts the encrypted values with the public key into plain text.
- 3 Displays the values in plain text on the user interface so that a user can modify it, if the correct password is provided.
- 4 Encrypts the plain text values with the environment-related keys.
- 5 Saves the encrypted values into the `ini` file of the target site.

BOX Manager global variables

For BOX Manager, global variables are defined as a type of BOX resource. They are displayed as a node on the BOX resource tree on each dialog box that displays BOX resources.

In this way, global variable type resources can be added/edited/removed/deployed in a uniform way similar to other types of BOX resources.

One BOX can only have one resource of this type for a site/mastersite, where the name is fixed and uneditable.

Note: You cannot manually add/remove global variable type resources in the **Edit BOX** dialog box and Create BOX wizard

In the **Deploy BOX** wizard, the **Resources** tab's **Configuration** tab for the Global Variable type resource node is read-only. This is similar to other resource nodes.

The **Merge BOX Resources into Current Site** page also shows the Global Variables node. Clicking the conflict indicator for the Variables node opens the **Resolve Conflict** dialog box. This has the option to add new and overwrite existing variables to the site (default), or add new variables to the site only.

Validation in other tools

The IDE tools have validation where global variables can be used, to report any incorrect global variable references.

For example, in NetConfig, when you click **Validate** or **Save**, the NetConfig file is read by the system as a plain text. It then uses pattern matching to find all possible strings, for example, `$$xxxxxxx`, to know which ones are not defined as global variables. It then prompts you with the invalid `$$xxxxxxx` list.

This validation is also found in Lookup Table Configurator, Alert Configurator, and Translation Configurator.

Tcl and Java interfaces for global variables

You can load global variables in Tcl UPoCs, Java UPoCs, and Java Driver.

Two Tcl commands are available. You can use these in all Tcl scripts that are used inside a site.

For additional information, see [Global variables commands](#) on page 1420.

The Java interface for Global Variable is provided by `$HCIR00T/bin/gvjni.dll` and `$HCIR00T/lib/java/gvjni.jar`.

In general, `gvjni.jar` is always in the classpath for Java Driver and Java UPOC by default. In case of accidents, you can add the `gvjni.jar` path to the `$CLASSPATH` environment to correct the command.

You can also use interfaces according to Javadoc `$HCIR00T/docs/GlobalVariableJniAPI/index.html` to load global variable values in Java code.

Global variables Tcl interface usage

Global variables can simplify development and deployment of Cloverleaf interfaces.

These variables are defined once in a Cloverleaf site and can be used multiple times in these configuration files within that Cloverleaf site:

- NetConfig protocol parameters. For example, port number, directory path, and web services URL.
- Xlate files. For example, constants in input fields.
- Lookup tables. For example, input or output values.
- Arguments to UPoCs
- Alert definition entries

This table shows the global variables Tcl interface usage:

Global variable command	Usage	Variable	Result
<code>gvsubstring stringvalue</code>	Substitutes the global variable by its value in the string.	<code>stringvalue</code> : The string includes global variables.	Returns the string with global variable substituted.
<code>gvsubfile inputfilename ?outputfilename?</code>	Substitutes the global variable by its value in the file.	<code>inputfilename</code> : The absolute path of the file that includes global variables. <code>outputfilename</code> : The absolute path of the file to save the result. If not given, then the result is written to the input file.	Returns empty if success or the error message if an error.
<code>gvaddvar varname varvalue ?isencrypted?</code>	Adds the variable name and value to the table in memory.	<code>varname</code> : The global variable name. <code>varvalue</code> : The global variable value. <code>isencrypted</code> : If the global variable value needs be encrypted. "1" is encrypted "0" or others is not encrypted. Default is 0.	Returns empty if success, or the error message if an error.
<code>gvdelvar varname</code>	Deletes a global variable by name.	<code>varname</code> : The global variable name.	Returns empty if success or the error message if error.

Global variable command	Usage	Variable	Result
<code>gvsetvar varname varvalue ?isencrypted?</code>	Sets the global variable value in memory.	<p><code>varname</code>: The global variable name.</p> <p><code>varvalue</code>: The global variable value.</p> <p><code>isencrypted</code>: If the global variable value requires encryption.</p> <p>"1" is encrypted</p> <p>"0" or "others" is not encrypted.</p> <p>Default is 0.</p>	Returns empty if success or the error message if an error.
<code>gvgetvar varname</code>	Gets the global variable value by name.	<code>varname</code> : The global variable name.	Returns the global variable value if success, or the error message if an error.
<code>gvprint</code>			Prints out the global variable table in memory.
<code>gvsave</code>	Saves the global variable table in memory to the global variable <code>ini</code> file.		Returns empty if success, or an error message if an error.

Global variable example

In this example, a Developer creates a generic ADT interface with a port number referenced by a global variable `$$prodadtport`. This variable is created in the Global Variables Configurator with a value of 5555.

The Developer also has a constant `$$facilityID` referenced as an input value to a COPY operation in a translation. This variable is created in the Global Variable Manager with a value of "Infor_NA".

When creating a BOX with global variables, the Developer creates a BOX containing the necessary objects for the ADT interface to run. The two global variables are included in the BOX.

An Implementer deploys the BOX at a customer site. The two global variables are appended to the site global variables, or updated if they already exist.

During BOX deployment, the Implementer has the option of changing the value of the global variables. The implementer changes `$$prodadtport` to 5566 and leaves the value of `$$facilityID` intact. The implementer saves this configuration and the interfaces are ready to run with the global variable settings.

HL7 Configurator

HL7 (Health Level 7) is a standard for electronic data interchange in healthcare environments, with an emphasis on hospitals. HL7 refers to the highest level of the OSI, or Open System Interconnection, model from the ISO, or International Standards Organization. In the OSI conceptual model, communications software and hardware functions are separated into seven layers, or levels. The HL7 standard is primarily focused on the issues that happen within the seventh layer, or application level.

HL7 does not specifically conform to ISO-defined elements of the OSI seventh level. It also does not specify a set of ISO approved specifications to occupy levels 1-6 under HL7's abstract message specifications. HL7 conforms to the conceptual definition of an application-to-application interface.

The HL7 standard addresses the interfaces among various systems that send or receive ADT (Admissions, Discharge, and Transfer) data, queries, orders, results, and billing data. For more information about this standard, refer to *Health Level Seven: An Application Protocol For Electronic Data Exchange In Healthcare Environments*, available from Health Level Seven, Inc.

Variants

Using the HL7 Configurator, you specify HL7 variants as necessary for a given connection or site.

By creating variants, you add vendor-specific data to the standard definitions. For example, you can create z segments or modify message definitions.

Variants are based on the standard message layouts.

Add or modify fields, segments, and messages within these specific variants.

Note: v3 CDA 1.0 and 2.0 do not automatically display as existing variants. CDA is an XML-based message and is supported by importing the schemas/DTDs with the XML Package Manager.

CCD support

For CCD support, these packages are provided:

- HL7_CDA_2.0-xsd
HL7 CDA R2
- HL7_CCD
HL7 CDA R2
CCD Schematron
- HITSP_C32 (version 2.5)
HITSP C32 (CDA R2 with extensions)
CCD Schematron
C32 Schematron

File locations:

- HL7 CDA R2: `$HCIR00T/formats/xml/HL7_CDA_2.0-xsd`

- HITSP C32 (version 2.5): `$HCIR00T/formats/xml/HITSP_C32`
This includes `xsd` and `ocm` files according to the HITSP C32 (version 2.5) standard.
- CCD Schematron provides validation files for HL7 CDA R2: `$HCIR00T/xslt/CDA_2.0`
- C32 Schematron provides validation files for HITSP_C32 (version 2.5): `$HCIR00T/xslt/HITSP_C32`

The system implements the Schematron validation language by XSLT framework. These Schematron are based on the XSLT 1.0 Standard.

HL7 standard

The HL7 message format defines the standard data field, segment, and message layouts that are used in hospital communication systems. A complete implementation of the HL7 specifications is included with Cloverleaf, which defines these standards.

If message layouts follow these HL7 standards exactly, then the message formats can be used as provided.

If message layouts deviate in any way from the HL7 standard, then message formats must be user-configured, or assigned an HL7 variant. Each HL7 message-format variant that is created has a unique, user-specified name.

Use the HL7 Configurator to configure the variant. This tool provides the editing and display capabilities for defining and modifying data fields and assembling the segments for specific message layouts.

HL7 files reside in `$HCISITEDIR/formats/`.

When configuring the variant, always start with an HL7 standard as a base. Then, add or modify those fields, segments, and messages that differ from the standard. This procedure minimizes the amount of time spent configuring the message format.

A segment is a logical grouping of data fields. Each segment is given a name. For example, an ADT (Admissions, Discharge, and Transfer) message might contain these segments:

- MSH (Message Header)
- EVN (Event Type)
- PID (Patient ID)
- PV1 (Patient Visit)

Each segment is identified by a unique three-character code known as the Segment ID.

When using an Xlate, most messages do not have EVN-1 filled. If EVN-1 is required, then it is automatically filled with the event type when using an Xlate. If the field is optional, then it is not filled.

The MSH segment has certain required fields. If these fields are not set in a translation, then the outgoing message is created with default values. For example:

- `0(0).MSH(0).00011` (or `00014` in 2.1): Processing ID: "P", where P is for production.
- `0(0).MSH(0).00012` (or `00015` in 2.1): Version ID: the output version (for example, 2.1, 2.2, and so on).
- `0(0).MSH(0).00009` (or `00012` in 2.1): Message Type: the output message type (for example, ORM, and so on).

If the interface requires these fields to be blank, then you can nullify them by copying `@null` to the field.

The HL7 v3 CDA version is an XML-based standard for the interchange of clinical documents. The CDA message definition is defined by schemas or DTDs.

Because CDA is an XML-based message, it is supported by importing the schemas/DTDs with the XML Package Manager.

HL7 paths

To define specific elements within a message, path naming can use the location of an element within an HL7 message. Path names are used to specify groups, repetition counts, segment IDs and fields.

This also applies to X12 Configurator, XML Package Manager, and UN/EDIFACT Configurator.

This is a sample HL7 message:

```
MSH|^~\&|HIS|||19930603090227|ADT^A03|19930603090227|P|2.1|||
```

This is a group of segment definitions:

```
MSH
EVN
{PID}
[PV1]
{[Z01]}
```

The first two segments, MSH and EVN, display exactly once. These are required.

The PID segment must display at least once, but more instances can be used. This segment is also required. The curly braces define this segment as repetitive.

The PV1 segment is an optional segment defined by the square brackets.

- These are all in one group.
- A 0-based numbering system is used.
- These segments are all in group 0.

The Z01 segment is an optional segment defined by the square brackets. One to many instances are permitted.

Because groups, segments, and fields can repeat, a 0-based instance (a repetition count) is included for each object in a field path that repeats.

For example:

```
0(0).MSH(0).00013
```

The repetition count displays in parentheses if the object is a repeating item. If it does not repeat, then an instance is not noted.

Message type field

This path specifies the **Message Type** field (ID 00013) in the MSH segment:

```
0(0).MSH(0).00013
```

Dots (.) separate path items. Individual items are:

- 0(0)
The first instance of the first segment group.
- MSH(0)
The first instance of the MSH segment in the group.
- 00013
Field number 00013. This number does not have a repetition count. If the last item in an HL7 object path does not have an explicit instance count specified, then (0) is assumed.

For example, in the ADT_A17 message definition:

```
MSH
EVN
{
  PID
  PV1
}
```

The first two segments, MSH and EVN, are required and display exactly once.

The next pair of segments, PID and PV1, must display at least once, but more instances of the pair are permitted.

The PID-PV1 pair is the second group in the message.

```
MSH      0(0).MSH
EVN      0(0).EVN
{
  PID      1(0).PID
  PV1      1(0).PV1
}
```

Two or more segments can be present in a repeating group, marked by curly braces, or in an optional group, marked by square brackets. In these cases, they must have a separate group number.

All the items within a pair of curly braces or square brackets are in the same group number.

The PID-PV1 pair is the second group in the message: 1(0).

Accessing the patient name field

To access the patient name field in the PID segment that describes the first patient in a message, use the 1(0).PID(0).00041 field path. In this path, 1 selects the PID-PV1 segment pair group and (0) selects the first repetition of that group.

To access the patient name field in the second instance of the `PID` segment group, use this field path:

```
1(1).PID.00041
MSH
EVN
{
  PID      1(0).PID.00041
  PV1
}
{
  PID      1(1).PID.00041
  PV1
}
```

This is a more complex example showing sub-groups:

```
MSH      0(0).MSH(0)
EVN      0(0).EVN(0)
{
  [
    NK1      1(n).0(0).NK1(0)
    GT1      1(n).0(0).GT1(0)
  ]
  PV1      1(n).1(0).PV1(0)
  BLG      1(n).1(0).BLG(0)
}
[DSC]     2(0).DSC(0)
```

This example shows the `NK1-GT1` and `PV1-BLG` sub-groups.

```
MSH      0(0).MSH(0)
EVN      0(0).EVN(0)
{
  [
    NK1      1(n).0(0).NK1(0)
    GT1      1(n).0(0).GT1(0)
  ]
  PV1      1(n).1(0).PV1(0)
  BLG      1(n).1(0).BLG(0)
}
[DSC]     2(0).DSC(0)
```

Even though the `DSC` is a segment alone, it is not considered a part of the above grouping, making it a group by itself.

Field specifiers are appended to paths to indicate specific field and subfield values.

For example:

```
1(1).0(0).NK1(0).00048.[1]
```

Messages

A message is a unit of data transferred between connection points. Each message has a message type that defines its purpose. For example, the `ADT` message type is used to transmit portions of a patient's `ADT` data. It is transmitted from one connection point, the data source, to another, the data sink.

A message consists of a group of segments in a defined sequence. A three-character segment code contained within each message identifies the segment's type.

New segments can be optionally added at the top or at the end of the message.

HL7 fields

Note: The standard HL7 base definition cannot be deleted.

A field represents a specific piece of data, such as patient name, address, or doctor ID, and has a specific set of attributes.

Add or modify fields within specific variants. Select an existing variant or create a new one.

Selecting a field

The list box in the Defined Fields pane contains every field defined in the selected variant, numerically sorted.

- 1 For an existing field, click the field name, or specify a search string in the search text box. Press **Enter** or click **Search** (binoculars).
- 2 For a new field, click **New** to open the **Create New Field** dialog box.
 - For **Field Number**, specify a five-digit number that is unique among all fields in the current variant. User-defined fields are usually numbered 90000-99999. The list box automatically loads the next available field number within that range.
 - For **Field Name**, specify a descriptive name for the field. Unique field names are not necessary. To change an existing field name, double-click the field name to open the **Edit Name** dialog box.

Defining a field

- 1 Click the field name in the **Defined Fields** pane. Use the **Definition of the Selected Field** pane to modify existing or configure new fields.
- 2 For **Type**, click the arrow to open a menu list of data types.
- 3 For **Length**, click the arrow buttons to select the maximum number of characters that one instance of the field can occupy in a message.

Include component and subcomponent separator characters in this calculation. Select **Unlimited** for an unlimited field length.

HL7 segments

A segment is a logical grouping of data fields. Each segment is identified by a unique three-character code known as the segment ID.

Segments of a message can be required or optional and can happen only once in a message, or can repeat.

Add or modify segments within specific variants. Select an existing variant or create a new one.

Note: The standard HL7 base definition cannot be deleted.

Selecting a segment

The list box in the Defined Segments pane contains every segment defined in the selected variant, alphanumerically sorted.

- 1 For an existing segment, click the segment name, or specify a search string in the search text box at the bottom of the pane. Press **Enter** or click **Search** (binoculars).
- 2 For a new segment, click **New**. This opens the **Create New Segment** dialog box.
- 3 Specify a **Segment Name** and **Segment Description**.
For **Segment Name**, specify a three-letter/number combination for the segment name.
User-defined segments should have three-character names beginning with the letter Z, for example, "ZNS."

Defining the segment

Use the Definition of Segment pane to modify existing or define new segments. Modify a segment definition by editing, deleting, or adding new fields to the segment.

The list box lists the fields included in the segment.

- 1 Click the segment name in the Defined Segments pane. The selected segment definition displays in the Definition of the Selected Segment pane.
- 2 Designate where to place the new fields by clicking the **Add/Paste New Fields** arrow and making a selection.
For example, to add a fields after or before a particular field, first click the field where the insertion is located. Then, click **After Selected Field(s)** or **Before Selected Field(s)**.
- 3 Click **Add** that is located below the Definition list. This opens the **Add Field to Segment** dialog box.
 - a Click **List** and double-click the field to add.
 - b Click **OK** to place the field on the list at the location that was chosen under **Add/Paste New Fields**.
- 4 If necessary, then use the editing tools by clicking the appropriate tool that is located below the **Definition of the Selected Segment** list box.
- 5 Specify if the field is required, optional, or repeating.
Repeats specifies the number of repeats. Select the check box and then click **Unlimited** or a specific number of times the field repeats.

HL7 messages

Messages are the units of data transferred between connection points. Each message has a message type that defines its purpose. In HL7, each message consists of any number of segments.

Add or modify messages within specific variants. Select an existing variant or create a new one.

Note: The standard HL7 base definition cannot be deleted.

Selecting a message

The list box in the Defined Messages pane contains every message defined in the selected variant, alphabetically sorted.

- 1 For an existing message, click the message name, or specify a string in the search text box at the bottom of the pane. Press **Enter** or click **Search** (binoculars).
- 2 For a new message, click **New**. This opens the **Create New Message** dialog box.
- 3 For **Message Name**, specify an alpha-numeric combination for the message name. Numbers cannot be used as the first character.
User-defined segments should have three-character names beginning with the letter Z, for example, "ZNS."

Defining the message

Use the Definition of Message pane to modify existing or define new messages. Modify a message layout by editing, deleting, or adding new segments to the message definition.

Note: The MSH segment cannot be removed from a message definition. MSH is required and must be the first segment in the list. When a message is created, it is automatically added as the first segment.

- 1 Click the message name in the **Defined Messages** pane. The selected message's segments display in the **Definition of the Selected Message** pane.
- 2 Designate where to place the new segments by clicking the **Add/Paste New Segments** arrow and making a selection.
For example, to add a segment or segments after or before a particular segment on the list, first click the segment where the insertion is located. Then, click **After Selected Segment(s)** or **Before Selected Segment(s)**.
- 3 Click the **Add** tool that is located below the Definition list. This opens the **Add Segment to Message** dialog box.
 - a Click **List** and double-click the segment to add.
 - b Click **OK** to place the segment on the list at the location that was chosen under **Add/Paste New Segments**.
- 4 If necessary, then use the editing tools by clicking the appropriate tool that is located below the Definition of the Selected Message panel.
- 5 Specify if the message's segments are optional or repeating by selecting a single segment or a segment group:
 - **Optional** specifies the segment's inclusion is optional.
 - **Repeats** specifies the segment is repeating.
 - Shift-click to select a group of segments from the current message definition.

HL7 format definition example

This example shows how to define an HL7 variant by creating a new variant containing a new field and segment. You start by naming the variant. Select **File > New > version**. The **Create New Variant** dialog box is displayed. Specify an appropriate name and click **OK**.

By default, after creating a variant you are working with that variant. If you exit the HL7 tool, then the next time you restart the tool you must specify the variant with which to work.

Defining a new HL7 field

This variant is only a copy of the standard HL7 version. To create the new field to add to this variant:

- 1 Click the **Fields** tab.
- 2 In the **Defined Fields** pane, click **New**.
- 3 Specify the **Field Number** and **Field Name**. User-defined fields are usually numbered 90000 and above.
- 4 Click **OK**.
- 5 In the **Definition of the Selected Field** pane, select the **Type** and **Length** for the new field.

Defining a new HL7 segment

- 1 Click the **Segments** tab. In this example, create a Z01 segment for holding allergy data.
- 2 In the **Defined Segments** pane, click **New**.
- 3 Specify the **Segment Name** and **Segment Description**. User-defined segments should have three-character names beginning with the letter Z.
- 4 Click **OK**.
- 5 Click **Add/Paste Places New Fields** to select the location to place the new field.
- 6 Click **Add** (Folder tool) in the **Definition of the Selected Segment** pane to add the new field to the segment.
- 7 Click **List** and double-click the new field.
- 8 Click **OK** in the **Add Field to Segment** dialog box.
- 9 Repeat Steps 5 to 8 until all necessary fields are added.
- 10 Select the new segment in the **Definition of the Selected Segment** pane.
- 11 Click **Required** or **Optional** to make this segment required or optional in the message layout.
- 12 Select the **Repeats** check box to specify whether the segments repeats.
If repeating, then click the arrows to select the number of times the segment repeats.

Modifying the base HL7 message layout

- 1 Click the **Messages** tab.

- 2 Click the message in which to place the new segments in the Defined Messages pane. Its segments display in the Definition of the Selected Message pane.
- 3 Click **Add/Paste New Segments** to select the location to place the new segment. Click **Add** (Folder tool) in the Definition of the Selected Message pane to add the new segment to the message.
- 4 Click **List** and double-click the new segment.
- 5 Click **OK** in the Add Segment to Message dialog box.
- 6 In the Definition of the Selected Message pane, select the segment. Then select **Optional** or **Repeats** to make this segment an optional or repeatable item in the message layout.
- 7 Save the file.
- 8 Use the Testing Tool to test the variant.

HPRIM Configurator

HPRIM makes it possible to exchange data among health care professionals. For example, medical labs, clinical services and hospitals facilities, blood center, radiology facilities. HPRIM plays the role of HL7 in France.

HPRIM messages are created to exchange data among health care professionals. Exchanged data can be medical, such as test results, or administrative and financial, such as identity elements, billing elements, or payment elements.

The HPRIM recommendation is based on the ASTM American standard for the message structure definition part. This is ASTM E12-38: version 4.2 of August 2nd 1991.

The codifications that can be used for the exam denomination are defined by the technical commission H.PR.I.M.

The message definition files that are used by HPRIM are similar to those used for HL7. HPRIM files are stored in `formats/hprim`.

These files define the format:

- `Datatypes`
Primitive data types for fields
- `Fields`
One line defining each field
- `Segments.idx`
One line summarizing each segment
- `Segments/*`
One file defining each segment. The file name is the segment identifier.
- `Messages.idx`
One line summarizing each message
- `Messages/*`

One file defining each message. The file name is the same as the message name.

Segment combining and splitting

The **A** segment is an addendum segment for the preceding segment. The maximum length of a segment, including the segment separator, is 220 characters. For any segment beyond this size, one or more **A** segments must be used. There are two situations where this restriction is relevant:

- During parsing, before translation, of an HPRIM message, each **A** segment is combined with its preceding segment.
- During encoding, after translation, of an HPRIM message, any segment exceeding the maximum length limit is split into as many **A** segments as necessary.

The **C** segment is a comment segment for the preceding segment, which can display at any level of the hierarchy. This segment is removed from message before translation. This is implemented in the `preProcessHMD` TCL procedure.

Note: Creating a **C** segment is not supported. To keep the comment, you must create a variant to work around.

Changing the segment length

To change the segment length, create a variant and modify the Tcl code in `%HCIR00T%/formats/hprim/hmd.tcl`.

The global value `STANDARDSEGLength` can be modified.

```
set STANDARDSEGLength 220
```

This can be changed to:

```
set STANDARDSEGLength custom_segment_length
```

To preserve the standard version, change the value of `STANDARDSEGLength` only for the variant you have created.

For example:

```
if{cequal $strVariant variant_name}{set STANDARDSEGLength new_value}
```

Encoding separators

To enable you to set encoding separators for outbound messages, elements, an `ENCODE` keyed list, are added into the `SEPCHARS` keyed list. This makes the new `SEPCHARS` look similar to:

```
{FIELD "XXX"}
{COMPONENT "XXX"}
{REPEAT "XXX"}
```

```
{ESCAPE "XXX"}
{SEGMENT "XXX"}
{DECIMAL "XXX"}
{SUBCOMPONENT "XXX"}
{ENCODE
    {SEGMENT "XXX"}
}
```

To change the segment separator of an outbound message, set the `ENCODE.SEGMENT` value. This is retrieved in the `postProcessHMD` and used to encode the message.

To set this, in an Xlate `CALL` action:

```
set sepchars [xpmmetaget $xlateId SEPCHARS] keylset sepchars
ENCODE.SEGMENT "\r\n"
xpmmetaset $xlateId SEPCHARS $sepchars
```

HPRIM fields

A field represents a specific piece of data, such as patient name, address, or doctor ID, and has a specific set of attributes.

Add or modify fields within specific variants. Select an existing variant or create a new one.

The list box in the Defined Fields pane contains every field defined in the selected variant, numerically sorted. Add or delete fields in this pane.

- For an existing field, click the field name, or specify a search string in the search text box at the bottom of the pane. Press **Enter** or click **Find** (binoculars).
- For a new field, click **New**.
 - For **Field Number**, specify a five-digit number that is unique among all fields in the current variant. For ease in identification, user-defined fields can be numbered 90000-99999.
 - For **Field Name**, specify a descriptive name for the field. Unique field names are not necessary.

Note: The standard HPRIM base definition cannot be deleted.

Defining a field

Use the Definition of Field pane to modify existing or define new fields.

- 1 Click the field name in the Defined Fields pane. The selected field's definition displays in the Definition of the Selected Field pane.
- 2 Edit the field's attributes as required.

For **Type**, click the arrow to open a list of data types.

For **Length**, click the arrow buttons to select the maximum number of characters that one instance of the field can occupy in a message. Include component and subcomponent separator characters in this calculation. Select **Unlimited** for an unlimited field length.

HPRIM data types

Data type	Description
CM: composite	A field that is a combination of other meaningful data fields.
CE: coded element	This data type transmits codes and the text that is associated with the code.
ID: coded value for HPRIM defined tables	The value of such a field follows the formatting rules for an ST field except that it is drawn from a table of legal values. There is an HPRIM table number associated with ID data types. Examples of ID fields include religion and sex.
NM: numeric	A number represented as a series of ASCII numeric characters consisting of an optional leading sign (+ or -), the digits, and an optional decimal point. In the absence of a sign, the number is assumed to be positive. If there is no decimal point, then the number is assumed to be an integer.
TX: text data	String data meant for user display on a terminal or printer.
CQ: composite quantity with units	Components: <quantity (NM)> ^ <units (CE)> In future versions, CQ fields should be avoided because the same data can usually be sent as two separate fields, one with the value and one with the units as a CE data type.
ST: string data	String data is left-justified with trailing blanks optional. Any displayable (printable) ASCII characters (hexadecimal values between 20 and 7E, inclusive, or ASCII decimal values between 32 and 126), except the defined delimiter characters.
AD: address	Components: <street address (ST)> ^ <other designation (ST)> ^ <city (ST)> ^ <state or province (ST)> ^ <zip or postal code (ST)> ^ <country (ID)> ^ <address type (ID)> ^ <other geographic designation (ST)>
TS: time stamp	Format: YYYY[MM[DD[HHMM[SS[.S[S[S[S]]]]]]]][/-ZZZZ]^<degree of precision> Contains the exact time of an event, including the date and time. The date portion of a time stamp follows the rules of a date field and the time portion follows the rules of a time field.

Data type	Description
TN: telephone number	<p>For use in the United States and conforming countries, the telephone number is always in the form Format: [NN] [(999)]999-9999[X99999][B99999][C any text]</p> <p>For use in other countries, the optional first two digits are the country code.</p>
PN: person name	<p>Components: <family name (ST)> ^ <given name (ST)> ^ <middle initial or name (ST)> ^ <suffix (for example, JR or III) (ST)> ^ <prefix (for example, DR) (ST)> ^ <degree (for example, MD) (ST)></p> <p>This data type includes multiple free text components. Each component is specified to be an HPRIM ST data type. The maximum length of a PN field is 48 characters including component separators. The sending system may send upper- and lowercase or all uppercase. The receiving system may convert to all uppercase if required.</p>

HPRIM segments

A segment is a logical grouping of data fields. Each segment is identified by a unique three-character code known as the segment ID.

Segments of a message can be required or optional and can happen only once in a message, or can repeat.

Add or modify fields within specific variants. Select an existing variant or create a new one.

Selecting a segment

The list box in the Defined Segments pane contains every segment defined in the selected variant, alphanumerically sorted.

Note: The standard HPRIM base definition cannot be deleted.

- For an existing segment, click the segment name, or specify a search string in the search text box at the bottom of the pane. Press **Enter** or click **Find** (binoculars).
- For a new segment, click **New**, then specify a **Segment Number** and **Segment Name**.

Defining the segment

Use the Definition of Segment pane to modify existing or define new segments. Modify a segment definition by editing, deleting, or adding new fields to the segment.

The list box lists the fields included in the segment.

- 1 Click the segment name in the Defined Segments pane. The selected segment definition displays in the Definition of the Selected Segment pane.
- 2 Designate where to place the new field by clicking the **Add/Paste New Fields** arrow and making a selection. For example, to add a field after or before a particular field, first click the field where the insertion is located. Then, click **After Selected Field(s)** or **Before Selected Field(s)**.
- 3 Then, click the **Add** tool that is located below the Definition list.
Use this dialog box to select the new fields. Click **List** and double-click the field to add.
- 4 If necessary, then use the editing tools by clicking the appropriate tool that is located below the Definition of the Selected Segment list box.
- 5 Specify if the field is required, optional, or repeating:
 - **Required** specifies the field's inclusion is required.
 - **Optional** specifies the field's inclusion is optional.
 - **Repeats** specifies the number of repeats. Select the check box and then click **Unlimited** or a specific number of times the field repeats.

HPRIM messages

Messages are the units of data transferred between connection points. Each message has a message type that defines its purpose. In HPRIM, each message consists of any number of segments.

Selecting a segment

The list box in the Defined Messages pane contains every message defined in the selected variant, alphabetically sorted.

Note: The standard HPRIM base definition cannot be deleted.

- For an existing message, click the message name or specify a search string in the search text box at the bottom of the pane. Press **Enter** or click **Find** (binoculars).
- For a new message, click **New** and specify a **Message Name** and **Message Description**.

Defining the message

Use the Definition of Message pane to modify existing or define new messages. Modify a message layout by editing, deleting, or adding new segments to the message definition.

New segments can be added at any location in the message, but not before the Header (H) or after the Tail (L). Segments added before the "H" or after the "L" receive an error when using the HPRIM Test tool. This is located at **File > Test**, or by clicking **Testing Tool**.

To modify and define a message:

- 1 Click the message name in the Defined Messages pane. The selected message's segments display in the Definition of the Selected Message pane.

- 2 Designate where to place the new segments by clicking the **Add/Paste New Segments** arrow and making a selection.
For example, to add a segment after or before a particular segment on the list, first click the segment where the insertion is located. Then, click **After Selected Segment(s)** or **Before Selected Segment(s)**.
- 3 Click the **Add** tool that is located below the Definition list.
Use this dialog box to select new segments. Click **List** and double-click the segment to add.
- 4 If necessary, then use the editing tools by clicking the appropriate tool that is located below the Definition of the Selected Message pane.
To modify existing segments:
 - For a single segment, click the segment to highlight it.
 - For a group of segments, click the first segment in the group and then shift-click the last segment. This selects the entire group.
- 5 Specify if the message's segments are optional or repeating.
 - **Optional** specifies the segment's inclusion is optional.
 - **Repeats** specifies the segment is repeating.

HPRIM definition example

This example shows how to define an HPRIM variant by creating a new variant containing a new field and segment.

You start by naming the variant. Select **File > New > version** in the HPRIM Configurator.

Specify an appropriate name and click **OK**.

Defining a new HPRIM field

This variant is only a copy of the standard HPRIM version. To create the new field to add to this variant:

- 1 Click the **Fields** tab.
- 2 In the Defined Fields pane, click **New**.
- 3 Specify the **Field Number** and **Field Name**. For ease in identification, user-defined fields can be numbered 90000 and above.
- 4 Click **OK**.
- 5 In the Definition of the Selected Field pane, select the **Type** and **Length** for the new field.

Defining a new HPRIM segment

- 1 Click the **Segments** tab.
- 2 In the Defined Segments pane, click **New**.
- 3 Specify the **Segment Number** and **Segment Name**.

- 4 Click **OK**.
- 5 Click **Add/Paste Places New Fields** to select the location to place the new field.
- 6 Click **Add** (Folder tool) in the Definition of the Selected Segment pane to add the new field to the segment.
- 7 Click **List** and double-click the new field.
- 8 Click **OK** in the **Add Field to Segment** dialog box.
- 9 Repeat Steps 5 to 8 until all necessary fields are added.
- 10 Select the new segment in the Definition of the Selected Segment pane.
- 11 Click **Required** or **Optional** to make this segment required or optional in the message layout.
- 12 Select the **Repeats** check box to specify whether the segments repeats.
If repeating, then click the arrows to select the number of times the segment repeats.

Modifying the base HPRIM message layout

- 1 Click the **Messages** tab.
- 2 Click the message in which to place the new segments in the Defined Messages pane. Its segments display in the Definition of the Selected Message pane.
- 3 Click **Add/Paste New Segments** to select the location to place the new segment.
- 4 Click **Add** (Folder tool) in the Definition of the Selected Message pane to add the new segment to the message.
- 5 Click **List** and double-click the new segment.
- 6 Click **OK** in the **Add Segment to Message** dialog box.
- 7 In the Definition of the Selected Message pane, select the segment. Then, select **Optional** or **Repeats** to make this segment an optional or repeatable item in the message layout.
- 8 Save the file.

HRL definition

HRL (Hierarchical Record Layout) supports these types of FRL or VRL records in a single message:

- Repeating records
- Groups of records
- Repeating groups of records
- Optional groups of records
- Nested groups of records

A field in a previous FRL/VRL record within the same HRL determines the type of FRL/VRL that follows. Therefore, variations of a record format that are based upon a field in that record are possible.

HRL files use the `.hrl` extension. These files reside in `$HCISITEDIR/formats/`.

The format of a HRL definition consists of a list of keyed list entries. Each keyed list entry, also referred to as a segment, has a type that identifies its purpose. Each HRL segment consists of one or more logical records that match VRL, FRL, or HRL definitions from another file.

- If the type is FRL or VRL, then that segment's field information is defined in a FRL or VRL file.
- If the type is HRL, then that segment's information is recursively defined in another HRL file.

To create a new HRL file, click **New** and then click **New Segment**. This places a new segment in the Layout pane. Configure as necessary in the Segment Properties pane.

To reconfigure an existing HRL file, Click **Open** and select the HRL file to edit. In the Layout pane, click the segment to reconfigure. Its current properties display in the Segment Properties pane. Edit as necessary.

Condition options

Select **Condition** to specify which field to use containing the specified value. This option indicates which HRL segments are ignored unless the selected field contains a particular value. Specify in **Field** an FRL/VRL field from a previous HRL segment within the same .hrl file.

The condition options determine the presence/absence of a segment using a field from a previous segment.

You can use a **Repeat While** `segmentName.fieldName` to determine if the message consists of a certain FRL.

You can use a **Condition** on `previousSegmentName.fieldName` to determine if a message consists of a certain FRL based on the existence of a previous segment.

Repeat options

To indicate the number of repeating groups:

- Specify the number of repeats with a literal number in the HRL definition, or with a field name of the previous segment. For example, `Segment.field1`. Specify the number from a numeric field in a previous FRL/VRL that is defined within the same HRL.
- Encase the repeating records within a beginning and ending string specified in the HRL.

Records repeat as long as a specified FRL/VRL field within the record matches a specified text.

For an HRL segment to be considered repeating, you must select one of the repeat options:

- **No Repeat** indicates no repeats for the current HRL segment.
- **Repeats** indicates the current HRL segment repeats. Specify in the adjacent text box the total number of repeats for the specified FRL, VRL, or another HRL file containing the format information for this HRL segment. The default is 1. This value can also be a field name of the previous segment. For example, `Segment.field1`.

The Repeats text box can be a number or a field name of the previous segment.

- **Repeat Block** specifies a particular block for repeating.

The value of **Begin** or **End** is compared to the test data to determine where to repeat.

For example, with a given HRL segment that uses BBB as the Begin and EEE as the End for repeating, and the test data is BBB111222333444555666777888EEE, then 111222333444555666777888 is the repeat block, which is parsed out according to the FRL file defined in the segment:

```
Field
  0: ch >111<
  1: ch >222<
Field1
  0: ch >333<
  1: ch >444<
---- End FRL Segment Segment(0) ----
---- Begin FRL Segment Segment(1) ----
Field
  0: ch >555<
  1: ch >666<
Field1
  0: ch >777<
  1: ch >888<
```

- **Repeat While** repeats a specified file until the contents of **Field** does not contain the value specified in **Value**. The value of **Field** is an FRL/VRL field in the current HRL segment.

Note: The GUI supports writing escape characters. For example, if you use "\x0a" in an HRL definition, the engine correctly escapes the character.

Use this format for the field name:

```
<VRL segment name>.field name
```

Example:

To reference the field name `Patient_ID` in segment `PID`, use `PID.Patient_ID`.

Any FRL/VRL/HRL segment in a HRL is not required. The **Repeat While** option checks the value from the first instance of the segment. If the first instance does not meet the condition, then the whole segment is skipped, and the segment is shown as empty during parsing.

Example:

An HRL is set to "Repeat While" on a five-character FRL segment `Test`, whose first character should be A:

```
1. Message: ABCDEAfgHjKmnL ...
Field Number: 1
Field Name: Segment_Test
--- Begin FRL Segment Segment_Test(0) ---
Length of Repeat Field: 5
Repeat Field Offset from Field: 0
Field_1: >A<
Field_2: >BCDE<
---- End FRL Segment Segment_Test(0) ----
--- Begin FRL Segment Segment_Test(1) ---
Length of Repeat Field: 5
Repeat Field Offset from Field: 5
Field_1: >A<
Field_2: >fghi<
---- End FRL Segment Segment_Test(1) ----
2. Message: ZBCDEzfgHjKmnL ...
Field Number: 1 (empty)
Field Name: Segment_Test
```

Using Repeat While example

In this example, an HL7 lab result message is being translated to a flat/text file.

- 1 Set up a VRL for each segment's information, naming the first field similar to **Segment_ID**.
Use a three (3) character-length field and populate the field with the appropriate segment value, for example, OBX). Then, add any other fields you must have for that segment.
- 2 In this example, the VRL is called "OBX." In the HRL Configurator, add the OBX and click **Repeat While**.
For **Field** specify **OBX.Segment_ID**.
For **Value** specify **OBX**.
- 3 Do this for each of the segments.
For groups of segments, for example, the OBX group, define one HRL for each group with its appropriate VRLs. Then, reference that HRL within another HRL.
For example, HRL ORU_R01 contains:

```
VRL - MSH
VRL - PID
HRL - ORC
      VRL ORC
      HRL OBR
            VRL OBR
            HRL OBX
                  VRL OBX
                  VRL NTE
```

Using Repeat Block example

In this example, the user has a file that has a tape header. This is followed by a company header of 189 characters.

Under the company segment there is a repeating detail record, followed by the next company, and so on.

What is required is to have it repeating to the end of the first company detail, repeating over multiple companies. That is, the header line is followed by an iteration of companies. Each company consists of a company line followed by an iteration of detail lines. In essence, an iteration within an iteration.

(These are all FRL fields.)

```
"TAPE HEADER"
"Company A "
"detail1 0"
"detail2 0"
.
.
.
"Company B"
"detail1 0"
...
```

To have it repeat over the companies, another HRL is required that contains the HRL already made.

```
Create HRL called Company:
1 * Name: comp_line
  * File Name: comp.frl
  * No Repeat
2 * Name detail_line
  * File Name: det.frl
  * Repeat While Field: detail_line.indicator_field
  Value: DETAIL
Create another HRL called Total:
1 * Name: header_line
  * File Name: head.frl
  * No Repeat
2 * Name Company_hrl
  * File Name: Company.hrl
  * Repeat While Field: Company_hrl.indicator_field
  Value: COMPANY
```

If you are creating precisely one message per company, then use TCL to first get rid of the header line. Then, split the rest of the message in separate messages, one message per company. You can then use the company HRL to process these separate messages.

Empty segment handling

This table shows the options for handling an empty VRL/FRL segment in HRL.

Option	Description
Fill Spaces/Add Terminator	For FRL, when the segment has no content, it fills a fixed length of spaces for the entire FRL segment. For VRL, when the segment has no content, it adds a VRL terminator for the VRL segment.
Active Null	When the segment has no content, it leaves the entire segment empty for both FRL and VRL.

Empty segment handling example 1

In this example, an HRL is composed of four segments:

- FRL1 (four chars): abcd
- FRL2 (four chars): This is empty.
- VRL1 (terminator \$): ijkl
- VRL2 (terminator \$): This is empty.

When **Fill Spaces/Add Terminator** is selected, the HRL message is:

```
abdc    ijkl$$
```

It also picks up the segment once before checking the **Repeat While** condition. This indicates there is always one occurrence of the segment.

When **Active Null** is selected, the message is:

```
abdcijkl$
```

It checks the **Repeat While** condition only when it attempts to parse the segment. In this case, the segment may not exist in the incoming message.

Empty segment handling example 2

In this example, an HRL is composed of four segments:

- FRL1 (four chars)
- FRL2 (four chars)
- VRL1 (terminator \$)
- VRL2 (terminator \$)

Repeat While is selected and configured using a **Field** of **FRL2** and a **Value** as **efgh**.

The incoming message is:

```
abcd   efghijkl$mnop$
```

When **Fill Spaces/Add Terminator** is selected, the parsing result is:

```
FRL1: >abcd<
FRL2.(0): >   <
FRL2.(1): >efgh<
VRL1: >ijkl<
VRL2: >mnop<
```

When **Active Null** is selected, the parsing result is:

```
FRL1: >abcd<
FRL2: (empty)
VRL1: >   efghijkl<
VRL2: >mnop<
```

Import Script

You can use an external editor to create new or open existing Tcl files. Then, import the modified or generated Tcl script files back to the IDE using the **Import Script** tool.

When you edit javascript, python, and java files with an editor, the scripts must be imported to CIS. These are supported by the Import Script tool.

These files are imported to the corresponding CIS folders:

- Javascript (.js) and python (.py) files are imported under %HCISITEDIR%/scripts.
- Java (.java) packages and classes (.class) are imported to %HCISITEDIR%/java_uccs.

See [External Editor and Import Script tool](#).

JSON Configurator

JSON (JavaScript Object Notation) is an open standard format that uses human-readable text to transmit data objects consisting of attribute-value pairs. JSON is a language-independent data format. It is the primary data format used for asynchronous browser/server communication.

For the IDE, JSON support gives you the ability to compose a JSON configuration and apply the configuration to the translation and testing tools. The JSON Schema Configurator reads/writes the JSON schema configuration instance from/to the server side.

For additional information, see <http://www.json.org>.

JSON supports:

- Defining and editing JSON layouts, similar to the HL7 variant builder.
- Bidirectional translation between JSON and other Cloverleaf supported message formats.
- GRM support.
- Testing Tool support for parse, translate, and route.
- BOX Manager, Access Control, Site Document, Version Control, and CLAPI.

For example, on the **Box Manager Property** dialog box's **Resources** tab, you can add the JSON schema configuration from the current and master sites into BOX entry.

A set of currently available FHIR resources is included with JSON definitions.

Information storage

JSON layouts and variants are stored in %HCISITEDIR%/formats/json for user-defined data models, and %HCIRROOT%/formats/json for standard data models that are shipped with Cloverleaf.

JSON definitions include a set of currently-available FHIR resources.

JSON definitions are also supported by version control.

The site level jrl file is located in %HCISITE%/formats/json/package name.

The root level jrl file and FHIR (DSTU2) specification are located in %HCIRROOT%/formats/json/fhir.

FHIR support

JSON is a lightweight data-interchange format leveraged by the FHIR (Fast Healthcare Interoperability Resources) standard. Support for JSON provides robust support for FHIR and enhances the range of systems that can be integrated using Cloverleaf Integration Services.

This support provides:

- The ability to define and edit JSON layouts, similar to HL7 variant builder.
- Bidirectional translation between JSON and other Cloverleaf supported message formats.
- GRM support.
- Testing Tool support for parse, translate, and route.

Cloverleaf provides pre-defined FHIR JSON schemas that are applied to the Translation Configurator, Testing Tool, and others.

FHIR requires the JSON data elements to follow the order set in the XML schema. The JSON data elements are still optional, but they are in an expected order.

Using JSON/XML RESTful, the Cloverleaf FHIR box contains working examples and documentation to connect with a FHIR server. Then, you can successfully exchange patient demographics as defined in the HL7 FHIR Connectathon Track 1.

See http://wiki.hl7.org/index.php?title=FHIR_Connectathon_7.

XML support for FHIR is also provided. The pre-compiled `xsd` files for FHIR are located in `$HCIR00T/formats/xml/FHIR_1.0.2-xsd`.

The FHIR JSON (DSTU2) definition files are under `$HCIR00T/formats/json/fhir_1.0.2`.

There is a BOX located at `$HCIR00T/CAA/ws/samples/FHIR_example.box` that demonstrates how FHIR JSON can work in Cloverleaf.

This includes examples of:

- Re-using the standard FHIR definition from the root level and customizing the variant in the site level.
- Using a translation to convert a HL7 message to a JSON message.
- Using CAA-WS to send the FHIR JSON request to a FHIR server.

Reference nodes

The JSON Configurator can load reference nodes. These are read-only.

Reference nodes are displayed in the JSON Configurator in gray color.

JSON translation support

JSON is fully implemented in the GRM.

JSON can be used as input, output, or both for a translation.

BULKCOPY and PATHCOPY are supported. Ordering is maintained if ordering is enabled.

All JSON elements nodes are supported in translation, including `anyOf` and `oneOf`.

Configuration modes "Type" and "Ref"

The JSON Configurator has new configuration items:

- **Type**
- **Ref**

These cannot be defined in the same node. A JSON node can support one of these properties.

If an existing JSON file has both properties, then after saving changes, the JSON configuration automatically adjusts the format.

If a node is **Ref**, then it should have no **Type** property. Or, **Type** is ignored. If **Type** and **Ref** conflict, then some tools identify the schema to be invalid.

The conversion format follows these rules.

- When **Customized** is selected, **Ref** properties are removed.
- When a JSON node selected **Ref** is selected, all other properties belonging to the current node are removed.

The IDE can only create a JSON schema using the `Type` or `$ref` node.

The engine can change the current logic to loading the `$ref` node before the `Type` node. This only happens when the schema has both `$ref` and `Type`.

If you must keep the same translation result, then the only way to do this is to delete the `$ref` node from your JSON schema.

Schema References With \$ref

The `$ref` keyword is used to reference a schema, and provides the ability to validate recursive structures through self-reference.

An object schema with a `$ref` property must be interpreted as a `$ref` reference. The value of the `$ref` property must be a URI reference. Resolved against the current URI base, it identifies the URI of a schema to use. All other properties in a `$ref` object are ignored.

Caveats:

- The URI is not a network locator, only an identifier. It is unnecessary for a schema to be down-loadable from the address if it is a network-addressable URL. Implementations should not assume they should perform a network operation when they encounter a network-addressable URI.
- A schema must not be run into an infinite loop against a schema. For example, if two schemas `"#alice"` and `"#bob"` both have an `allof` property that refers to the other, a built-in validator might get stuck in an infinite recursive loop trying to validate the instance.
- Schemas should not make use of infinite recursive nesting.

Here is an example of a JSON schema validator tool. If the type of `$ref` field does not match with `type` field, then the validator identifies the schema as invalid.

Schema:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "description": "Modified JSON Schema draft v4 that includes the optional '$ref' and 'format'",
```



```

"definitions": {
  "positiveInteger": {
    "type": "integer",
    "minimum": 0
  },
  "type": "object",
  "properties": {
    "maxLength": {
      "type": "integer",
      "$ref": "#/definitions/positiveInteger" }
  }
}
Data:
{
  "maxLength": "A36697"
}

```

Combining schemas

JSON schema includes keywords for combining schemas. This is useful when using a value to simultaneously validate against multiple criteria. These keywords are mostly used with `$ref` nodes.

Combined schemas are displayed as a combined data structure on the Translation Configurator JSON tree.

Because the parent can have the same name as the children, it is difficult to know which child belongs to which parent.

To eliminate name collision that can happen if multiple child schemas share an element name, the parent-child relationship is enforced on these type nodes by displaying individual nodes.

If the individual node is a reference, then the indexes under these nodes are named with the definition name; otherwise, the node uses the index name.

The index node name is added to the translation source and destination address using the “.” separator.

You can use these keywords to combine schemas:

Keyword	Condition
<code>allOf</code>	Must be valid against all of the sub-schemas
<code>anyOf</code>	Must be valid against any of the sub-schemas
<code>oneOf</code>	Must be valid against exactly one of the sub-schemas

Using list and tuple for array

When array is selected for **Type**, the **Element Mode** list has two options:

- list: All elements in **array** are the same definition.
The items that are defined for **list** apply to all elements that are under this array node in a JSON message.

Example definition:

```
{
  "type": "array",
  "items": {
    "type": "number"
  }
}
```

A correct message is: [1, 2, 3, 4, 5]

An incorrect message is: [1, 2, "3", 4, 5]

- tuple: Each element in **array** can be another definition.

You must define an item under tuple for each element under this array node in a JSON message. The order should be the same.

Example definition:

```
{
  "type": "array",
  "items": [
    { "type": "number" },
    { "type": "string" },
    { "type": "string" },
    { "type": "string" }
  ]
}
```

A correct message is: [1600, "Pennsylvania", "Avenue", "NW"]

An incorrect message is: ["Pennsylvania "] or ["Sussex", 24, "Drive"] (The first element should be a number.)

Required, Validation, and Ordering options

Set the node to **Required** if a node must display in the JSON message.

Clear **Validation** on the JSON definition root node if you do not require Cloverleaf to validate the JSON message against the definition.

Example definition:

```
{
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "id": { "type": "string" },
    "address": { "type": "string" }
  }
}
```

Data:

```
{"name": "Mike", "address": "XXXX", "id": 12345}
```

In this example, incorrect usage is when **Validation** is selected. "id" should be a string, not a number.

For example, each JSON node content must be validated by Cloverleaf and checked whether the nodes display in the same order as the defined definition. To do this, select **Ordering** on the JSON definition root node.

Example definition:

```
{
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "id": { "type": "string" },
    "address": { "type": "string" }
  }
}
```

Data:

```
{"name": "Mike", "address": "XXXX", "id": "12345"}
```

In this example, incorrect usage is when **Ordering** is selected ("id" should be between "name" and "address.")

Undefined nodes in the JSON message causes the ordering check to fail even if **Additional Properties** is enabled.

If undefined nodes are used in a JSON message, then select **Additional Properties** for object, or **Additional Items** for array.

Additional Items in array does not effect ordering.

You can define the basic elements in a separate file, after which these elements can be referred to by other JSON definitions.

The element should be defined in the definitions section.

The referring node and referred node should have the same type.

The main JSON definition and reference files should be in the same package.

Ordering and Outbound Ordering

When the JSON Configurator is opened, the JSON Root node is displayed. When you select "JSON Root," **Outbound Ordering** is displayed as an option.

Note: Although the option name is "**Outbound Ordering**" on the user interface, in the schema file it is named "orderEncoding".

When **Ordering** and **Validation** are enabled, there is basic validation and advanced validation (order validation) for input messages.

When **Ordering** is enabled, the output message keeps the order as a schema definition.

When **Outbound Ordering** and **Validation** are enabled, there is only basic validation for input messages.

When **Outbound Ordering** is enabled, the output message keeps the order as a schema definition.

Configuring JSON in Cloverleaf

The JSON definition is used to describe the JSON message data. Cloverleaf uses the JSON definition to validate the JSON message and perform translation on the JSON message. You must configure the JSON definition before the JSON message can be handled in Cloverleaf.

JSON schema is used as the rule for the Cloverleaf JSON definition.

For additional information, see <http://json-schema.org>.

The order of configuration is:

- 1 Create the JSON definition by basing it on a JSON message or by manually editing nodes.
See [Creating a JSON definition by manually editing nodes](#) or [Creating a JSON definition based on a JSON message](#).
- 2 Test the JSON message.
See [Testing the JSON message against a definition](#).
- 3 Configure the translation.
See [Configuring a translation for the JSON format](#).
- 4 Determine the Trx ID.
See [Determining the Trx ID on a JSON message](#).

Creating a JSON definition by manually editing nodes

To configure a general JSON definition, configure the JSON definition in the JSON Configurator by adding each node to the root node.

- 1 Open the IDE and select the JSON Configurator.
- 2 Select the root node and add a sub-node under it.
- 3 Set the properties for that node.
- 4 Edit other nodes as required.

Creating a JSON definition based on a JSON message

When you use a JSON message, Cloverleaf automatically creates the JSON definition based on that message. To do this:

- 1 Prepare the JSON message.
- 2 Open the IDE and select the JSON Configurator.
- 3 Click **Load**, then import the JSON message.
- 4 Manually change any node as required.
- 5 If undefined nodes are used in a JSON message, then select **Additional Properties** for object, or **Additional Items** for array.

- 6 For additional configuration, you can set the node to **Required** if a node must display in the JSON message. You can also clear **Validation** on the JSON definition root node if you do not require Cloverleaf to validate the JSON message against the definition.

See [Required, Validation, and Ordering options](#).

Testing the JSON message against a definition

After the JSON definition is configured, use the JSON testing tool to check whether the definition is correct.

The testing tool can also be used to check whether the JSON message matches the definition.

Example data:

```
{
  "Name": "Mike",
  "ID": "12345",
  "Contacts": [
    { "Type": "Home", "Address": "XXX", "Telephone": ["000", "12345678"] },
    { "Type": "Work", "Address": "YYY", "Telephone": ["111", "87654321"] }
  ],
  "Details": {
    "Sex": "Male",
    "Age": 40
  }
}
```

Parse result:

```
Name >Mike<
ID >12345<
Contacts(0).Type >Home<
Contacts(0).Address >XXX<
Contacts(0).Telephone(0) >000<
Contacts(0).Telephone(1) >12345678<
Contacts(1).Type >Work<
Contacts(1).Address >YYY<
Contacts(1).Telephone(0) >111<
Contacts(1).Telephone(1) >87654321<
Details.Sex >Male<
Details.Age >40<
```

Contacts and Telephone are arrays, so they have an index number after them to indicate the specific element in the JSON message.

Configuring a translation for the JSON format

The JSON format can be used in the Translation Configurator. You must prepare your JSON definitions before configuring a translation file on the JSON format,

When the action works on a specific element of an array node, enter the index number in the `()` after that node. For example, `Contacts(0).Telephone(1)`. This is the second element of Telephone in the first element of Contact.

For ITERATE, Cloverleaf only supports the ITERATE action on an array node. In this case, the basis should be an array node. There is no difference between field, group, and segment under **Type**. You can select any of these types and use the variable in the address.

For ITERATE on an array-root JSON message, you can leave **Basis** empty. In this case, the basis is the array root. For example:

```
[
  { "Type": "Home", "Address": "XXX", "Telephone": ["000","12345678"] },
  { "Type": "Work", "Address": "YYY", "Telephone": ["111","87654321"] }
]
```

COPY works only on basic type nodes. For example, string, number, integer, boolean, and null. For array and object, use PathCopy to copy the entire node and its child nodes.

JSON use cases

These use cases cover:

- Raw routing a JSON message from one system to another.
- Storing a JSON transaction in SMAT.
- Delivering the translated JSON to a remote endpoint.

Use case 1: To raw route a JSON message from one system to another

- 1 Create two threads, `json_in` and `json_out`.
- 2 Configure the protocol properties of each thread to comply with each JSON endpoint.
- 3 Configure **JSON** as Trx ID determination. Alternatively, you can select UPoC for custom Trx ID on the `json_in` thread.
- 4 Configure a static route with Raw translation between `json_in` and `json_out`.

Use case 2: Storing a JSON transaction in SMAT

Enable inbound and outbound saved messages for the `json_in` thread that was created in the first use case. The JSON data is then written to the SMAT archive.

Use case 3: Delivering translated JSON to a remote endpoint

To translate a JSON message from one system to another:

- 1 Create two JSON message models using the JSON record layout tool:
 - `json_in.json`
 - `json_out2.json`
- 2 Create a translation called `json.xlt`. Configure the translation to use `json_in.json` as the input record layout and `json_out2.json` as the output record layout.
- 3 Configure the translation to convert the data from input to output. Operations such as BULKCOPY and PATHCOPY are supported.
- 4 Create a new thread `json_out2`.

- 5 Configure the protocol properties of `json_out2` to comply with the endpoint.
- 6 Add a new route to the `json_in` thread, routing all data to `json_out2` using the `json.xlt` translation.

Determining the Trx ID on a JSON message

There are two methods to get the Trx ID from a JSON message. For **Trx ID Determination Format** you can select **JSON** or **Field Routing**.

When **Field Routing** is selected, choose a JSON definition and select the nodes whose values are picked up as Trx ID.

When **JSON** is selected, Cloverleaf always gets the first type node value as the Trx ID. For example:

Data:

```
{
  "Name": "Mike",
  "ID": "12345",
  "Contacts": [
    { "Type": "Home", "Address": "XXX", "Telephone": ["000", "12345678"] },
    { "Type": "Work", "Address": "YYY", "Telephone": ["111", "87654321"] }
  ],
  "Details": {
    "Sex": "Male",
    "Age": 40
  }
}
```

Trx ID: Mike

Data:

```
{
  "Contacts": [
    { "Type": "Home", "Address": "XXX", "Telephone": ["000", "12345678"] },
    { "Type": "Work", "Address": "YYY", "Telephone": ["111", "87654321"] }
  ],
  "Name": "Mike",
  "ID": "12345",
  "Details": {
    "Sex": "Male",
    "Age": 40
  }
}
```

JSON Schema Configurator

JSON schema is used to describe the structure of JSON data. This is written in JSON.

"schema" keyword

The `$schema` keyword is used to declare that a JSON fragment is a piece of JSON schema. It also declares which version of the JSON schema standard that the schema was written against. We recommend that all JSON schemas have a `$schema` entry.

Some files do not contain the `$schema` keyword. These are treated as a JSON data file and displayed in a from the JSON schema file in JSON Configurator.

The `$schema` keyword check has been removed to support the schema files that are missing the `$schema` field. This displays as a JSON schema file. Displaying the JSON data file is no longer supported in JSON Configurator.

Note: The JSON Schema Configurator does not check the **\$schema** field. The configurator treats a JSON file as schema by default, so it cannot display the JSON data file in the configurator. It only processes schemas.

"definition" node

The `definition` node is useful for structuring the complex schema into parts. These parts can be reused in other places. They can refer to this schema snippet from other locations using the `$ref` keyword.

The JSON Schema Configurator shows `$ref` and `definition` on the tree to aid in understanding the JSON node structure.

The Site Documentation displays the `definition` and `$ref` nodes. Reference node can be expanded. If there is an infinite loop, then only limited levels are expanded.

JSON supports reference to sub-elements of definition

A JSON reference is a JSON object that contains a member named `"$ref."` This member has a JSON string value.

Example:

```
{ "$ref": "http://example.com/example.json#/foo/bar" }
```

Reference to any other elements using URI reference is supported.

Note: This feature is supported on engine runtime but not on the GUI. You can input a reference, for example, `Account.schema.json#/definitions/Account/extension`, and save it, although the GUI incorrectly displays it.

JSON Schema Configurator user interface

The JSON Schema Configurator displays `$ref`, `definition`, and other nodes on the node tree, if they exist. The node order respects the order in the `json` file.

Similar to `properties` of `type` object, `definition` is not displayed as a node on the tree. All definitions are displayed directly under the root-level node.

Instead of `JSON Root`, the root node is the json schema file name without the `.json` suffix. If this is a new file, then the root node is `untitled` and only has the `schema` property.

The `JSON Root` node is under the top-level node, and is created by default. The properties are the same as the previous `JSON Root` node.

New schema files include:

- Basic schema file that contains no `definition`, and `reference`, that only has some fields:

```
JSON schema file name
JSON Root
  Field 1
    Subfield 11
    Subfield 12
  Field 2
  Field 3
```

- Schema file that contains `definition` and some fields:

```
JSON schema file name
Definition 1
Definition 2
JSON Root
  Field 1
  Field 2
  Field 3
```

- Schema file that contains `$ref` and `definition`:

```
JSON schema file name
JSON Root ( This is a reference node )
  Field 01
  Field 02
  Reference 01
    Reference 011
    Field 011
    Field 012
  Field 03
Definition 1
  Reference 11
    Field 11
    Field 12
Definition 2
  Field 21
  Field 22
```

The structure of an individual definition, field, and reference are shown in the same configuration.

The reference node, definition node, and field node are displayed with different icons.

JSON schema definitions

The JSON Schema Configurator supports displaying `definition` and adding, modifying, and deleting `definition`.

Definitions are stored in the `json` file, which respects the JSON schema standard.

Add definition

Right-clicking a top-level node opens the **Add Definition** menu. The property panel's title is **Definition Property**.

`definition` has properties similar to other fields, except for the configuration mode, which is customized.

Modify or delete definition

`definition` and its sub-nodes can be edited. `definition` can also be deleted.

A warning is triggered when the definition name has changed and applied. This happens when there is a risk of it being referenced by other schemas when clicking another node on the JSON tree or saving the JSON file. This also applies to the **Delete** action. A **Confirm** dialog box opens when `definition` is cut or deleted, since it could be referenced by other schemas.

JSON Pointer

The JSON Pointer is a string syntax for identifying a specific value within a JSON text document. This syntax can be expressed in JSON string values and Uniform Resource Identifier (URI) fragment identifiers.

The `$ref` value respects JSON Pointer. For example, this is supported by the GUI: `/fhir_1.0.2/bundle.schema.json#/definitions/Bundle_Entry/properties/link`.

Example:

- 1 Open JSON Configurator. Under **Create new file**, select "New field".
- 2 Select **Reference mode**.
- 3 Specify `/fhir_1.0.2/bundle.schema.json#/definitions/Bundle_Entry/properties/link`. This loads the node structure on the JSON tree.

Note: In CIS 20.1, the engine does not support JSON Pointer, but does support JSON Path. Because of this, when `$ref` has a number path, for example, `Bundle.schema.json#/definitions/Bundle/allOf[1]/properties/link`, the path cannot be used in `$ref`. In another example, you can input a reference, for example, `Account.schema.json#/definitions/Account/extension`, and save it. However, the GUI incorrectly displays it.

Syntax

A JSON Pointer is a [Unicode] string containing a sequence of zero or more reference tokens, each prefixed by a `"/"` (`%x2F`) character.

If a reference token contains `"/"` (`%x2F`) or `"\"` (`%x5C`) character, then these characters must each be prefixed (escaped) with a `"\"` (`%x5C`) character.

ABNF syntax:

```
json-pointer = *( "/" reference-token )
reference-token = *( unescaped / escaped )
unescaped = %x00-2E / %x30-5B / %x5D-10FFFF
escaped = "\" ( "/" / "\" )
```

It is an error condition if a JSON Pointer value does not conform to this syntax.

Evaluation

Evaluation of a JSON Pointer begins with a reference to the root value of a JSON text document. It completes with a reference to another value within the document. Each reference token in the JSON Pointer is sequentially evaluated.

Evaluation of each reference token begins by unescaping any escaped character sequence; this is performed by the "\" (escape) prefix. The reference token then modifies which value is referenced according to one of these schemes:

- If the currently referenced value is a JSON object, then the new referenced value is the object member with the name identified by the reference token. If a referenced member name is not unique, then referenced member is undefined.
- If the currently referenced value is a JSON array, then the reference token must contain an unsigned base-10 integer value. The new referenced value is the array element with the zero-based index identified by the token.
- If a reference token is being evaluated against a concrete JSON document, then the implementation might evaluate each token against a concrete value. The evaluation is terminated with an error condition if a evaluation fails to resolve a concrete value.

JSON String representation

A JSON Pointer can be represented in a JSON string value. All instances of quotation mark "\"" (%x22), reverse solidus (backslash) "\" (%x5C), and control (%x00-1F) characters must be escaped.

JSON schema reference

You can modify the `reference` JSON node URL. You can expand this in the JSON tree as a read-only node after the reference node has successfully loaded from the server side.

Changes to `reference` are validated on the user interface and server sides.

`reference` can be expanded in the JSON tree as a read-only node. When you right-click a reference node, a dialog box opens with **Copy**, **Delete**, and **Move down** options.

After you edit the `ref` value, the **JSON Schema Configurator** verifies the `ref` value with these methods:

Client-side validation

On the client side, the `ref` value must contain at least one "#" followed by a definition name.

An `Invalid ref!` error message opens to alert you when the `ref` value does not apply the rule. The cursor then moves to the `ref` value's text field.

Server-side validation

Server-side validation goes through these steps:

- 1 Search the reference file from the local site, master site, and root level. The local file URI is supported in the JSON Schema Configurator.

- If the configuration begins with “/”, then search the file starting from \$HCIR00T\formats\json.
- If the `ref` value does not begin with “/”, then search the file starting from \$HCIR00T\formats\json\
Package name.

If the file cannot be found, then a `Cannot find xxxx reference file!` warning message opens.

2 Validate the `ref` node, that it is in the reference file.

If the node is not found in the reference file, then a `Cannot find xxx definition in yyyy file!` warning message opens..

When the user opens a JSON schema file, the same validation applies to the `ref` value.

Saving the configuration

The JSON Schema configurator only saves the `ref` node configuration.

This structure displays on the JSON tree:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "ordering": false,
  "validationcheck": true,
  "properties": {
    "REF": {
      "$schema": "http://json-schema.org/draft-04/schema#",
      "ordering": false,
      "validationcheck": true,
      "$ref": "p1/p2/Ref_nest1.json#/definitions/nest1_0"
    }
  }
}
```

Loading JSON schema

- 1 In the JSON Schema Configurator main panel, change the menu's **loading . . .** name to **Load JSON Schema . . .**
- 2 On the toolbar, change the icon's tooltip from `loading . . .` to `Load JSON Schema . . .`
- 3 On the toolbar, clicking the **Load JSON Schema** icon in the toolbar. This opens a file chooser dialog box, where you can select a JSON schema file on the client side.

Converting JSON data to JSON schema

- 1 Before converting a data file to a schema file, ensure there is no `$Schema` file in the JSON file; otherwise, an `Invalid JSON data file!` error message opens.
- 2 In the JSON Schema Configurator main panel, select **File > Convert to JSON Schema**. This places a new load file icon on the toolbar.
- 3 Select **Convert to JSON Schema**. This opens a file chooser dialog box for selecting a client-side JSON data file.

LDL Configurator

DICOM contains length-delimited fields. The LDL (Length Delimited Layout) tool specifies a DICOM record layout. Standard DICOM transactions are available in the root and can be edited with the LDL tool.

The LDL Configurator supports explicit length fields, undefined length fields, and sequenced fields.

DICOM is supported as DICOM Router. It receives the DICOM file as DICOM SCP (DICOM Service Class Provider). Then, it routes the DICOM file after transformation to another DICOM SCU (DICOM Service Class User). Conversion from DICOM to other message formats, for example, HL7, is also possible.

Parts of DICOM include:

- **DICOM format**
A DICOM object is a list of DICOM elements. Each DICOM element is an LDL record of the format (TAGLENGTH, DATA).
TAG is composed of two short integers, indicating (GROUP, , ELEMENT) and implicitly defines the type for the **DATA** field.
- **DICOM element**
DICOM elements are one of 28 types, also called VR, or value representation.
DICOM elements are nested under a parent element as the SEQUENCE DICOM element (SQ). This organizes DICOM in a hierarchical structure.
- **DICOM group**
A GROUP is a logical grouping of data elements, for example, patient identification group or a CT scan group.
The ELEMENT, is the individual item within that group, for example, patient ID, patient name, or a pixel data element.
An even numbered GROUP OR ELEMENT is DICOM-specified and odd numbered are user-specified. This is similar to Z segments in HL7.
An element that defines the GROUP always has 0x0000 as its ELEMENT.
- **DICOM SOP**
DICOM SOP (Service Object Pairs) are DICOM objects that trigger running a service and contain service-specific information.
The DICOM SOP is composed of two parts:
 - DIMSE (Dicom Message Service Element) that contains command information.
 - IOD (Information Object for Dicom) that contains data for that command.
 DIMSE is a list of ELEMENTS under GROUP 0x0000 and is indicated by two elements:
 - Service Class (0x0000, 0x0002)
 - Command Code (0x0000, 0x0100)
 A SOP might not have any data, thus it solely consists of DIMSE. If the SOP requires data to be sent, then the presence of data is indicated by setting element (0x0000, 0x0800) to NULL (=0). If set to NULL, then it is expected that there is data following when GROUP 0x0000 ends.
DICOM SOP are service-specific and contain structure and data specific to the service being requested. The DICOM services are also called DIMSE Services, since it is the DIMSE component of SOP that identifies the service.

For more details, see the DICOM dictionary at <http://rsb.info.nih.gov/nih-image/download/documents/dicom-dict.txt>.

Additional points:

- DICOM fields are always even byte lengths, so fields are padded to make them even. Conversion to and from the DICOM format should be performed in a similar manner.
- Implicit versus explicit DICOM element format should be understood before conversion.
- Groups may have zero lengths. In DICOM, this is available only for groups.
- DICOM GROUP and ELEMENT are always in ascending order of their numerical value.
- An element can have multiple values. For fixed length elements, such as numerical values, the element length is in multiples to indicate this fact. For variable length elements, for example, text strings, a backslash “\” is used to delimit sub elements.
- PDU numeric data fields are always Big Endian.

LDL definition files

The LDL definition files are located under `$HCIR00T/formats/ldl`.

The root-level definition provides standard datatypes, fields, and C/N services DIMSE header segment files.

You must define a site-level DATASET segment file for the actual `dicom` file.

This folder contains these files:

- `datatypes`: Contains definitions from DICOM Spec Part 5 Section 6.2.
- `fields`: Contains definitions from DICOM Spec Part 6 Section 6 & DICOM Spec Part 7 Annex E.
- `messages.idx`: Describes the entries in the messages folder.
- `segments.idx`: Describes the entries in the segments folder.
- `messages`: Contains definitions for the DICOM message, including DIMSE and DATASET.
- `segments`: Contains definitions from DICOM Spec Part 7.

Configuring an LDL field

The list box in the Defined Fields pane contains every field defined in the selected variant, numerically sorted. Add or delete fields in this pane.

Note: The standard LDL base definition cannot be deleted.

- 1 For an existing field, click the field name or specify a search string in the **Search** field. Click **Search** (binoculars).
- 2 To create a new field, click **New**. This opens the **Create New Field** dialog box.
 - a For the **Field Number**, specify a number that is unique among all fields in the current variant. Field numbers have two sections: GGGG,XXXX. The GGGG section should be odd and not 0001, 0003, 0005, or 0007. The XXXX section should not be in the range of 0001-000F or 0100-0FFF.
 - b For the **Field Name**, specify a descriptive name for the field. Unique field names are not necessary.

- c Click **OK**. The new field displays in the **Defined Fields** list box.
- 3 To change a field name, double-click a user-defined field name in the list. This opens the **Edit Name** dialog box.
- 4 Specify the new name and click **OK**.

Defining the field

Use the **Definition of Field** pane to modify existing or define new fields.

- 1 Click the field name in the **Defined Fields** pane.
The selected field's definition displays in the **Definition of the Selected Field** pane.
- 2 Edit the field's attributes as required.
 - **Type:** Click the arrow to open a list of data types. Different types have a defined length.
 - **Value Multiple:** This indicates that fields can have at most multiple times of this length. For example, a field is type SS and has a basic length of "2". If the **Value Multiple** is defined as "2", then the field content can be "4". When the engine shows the content, it is "123\456".

Using retired fields

The root-level LDL fields that are provided with this release do not include retired fields according to the 2014 specification.

Retired fields are manually added back through the IDE. These fields are saved at the site-level.

- 1 Click **New** to open the **Create New Field** dialog box.
- 2 Select **Add a retired field**.
This opens an information area and enables **List** for **Field Number**.
- 3 Click **List** to open the **Field Number** dialog box.
If a retired field has the wildcard character "x", then you must replace the "x" with a character between 0 to 9 or A to F.
- 4 Select a field and click **Apply**.
This populates the **Create New Field** dialog box.
- 5 Click **OK** to add it to the **Defined Fields** panel.

Configuring an LDL segment

A segment is a logical grouping of data fields. Each segment is identified by a unique name.

Segments of a message can be required or optional and can happen only once in a message, or can repeat.

The list box in the Defined Segments pane contains every segment defined in the selected variant, alphanumerically sorted.

Note: The standard LDL base definition cannot be deleted.

- 1 For an existing segment, click the segment name or specify a search string in the Search field. Click **Search** (binoculars).
- 2 To create a new segment, click **New**. This opens the **Create New Segment** dialog box.
 - a Specify a **Segment Name** and **Segment Description**.
 - b Click **OK**. The new segment displays in the **Defined Segments** panel.

Defining the segment

Use the Definition of the Selected Segment pane to modify existing or define new segments. Modify a segment definition by editing, deleting, or adding new fields to the segment.

The list box lists the fields included in the segment.

- 1 Click the segment name in the **Defined Segments** pane. The selected segment definition displays in the Definition of the Selected Segment pane.
- 2 In the **Definition of the Selected Segment** pane, designate where to place the new field by clicking the **Add/Paste New Fields** arrow and making a selection.
- 3 Then, click the **Add** tool that is located below the Definition list. The **Add Field to Segment** dialog box is displayed.
- 4 Click **List** to locate the field to add.
- 5 Select a field and click **Apply**.
- 6 Click **OK** on the **Add Field to Segment** dialog box.

Configuring an LDL message

Messages are the units of data transferred between connection points. Each message has a message type that defines its purpose. In LDL, each message consists of any number of segments.

Note: The standard LDL format does not include messages, although a variant of LDL should define some messages. The standard LDL base definition cannot be deleted.

Add or modify messages within specific variants. Select an existing variant or create a new one.

- 1 For an existing message, click the message name or specify a search string in the **Search** field. Click **Find** (binoculars).
- 2 To create a new message, click **New**. This opens the **Create New Message** dialog box.
 - a Specify a **Message Name** and **Message Description**.
 - b Click **OK**. The new message displays in the **Defined Messages** panel.

Defining the message

Use the **Definition of the Selected Message** pane to modify existing or define new messages. Modify a message layout by editing, deleting, or adding new segments to the message definition.

The list box in the **Defined Messages** pane contains every message defined in the selected variant, alphabetically sorted.

- 1 Click the message name in the **Defined Messages** pane. The selected message definition displays in the **Definition of the Selected Message** pane.
- 2 In the **Definition of the Selected Message** pane, designate where to place the new segment by clicking the **Add/Paste New Segments** arrow and making a selection.
- 3 Then, click the **Add** tool that is located below the Definition list. The **Add Segment to Message** dialog box is displayed.
- 4 Click **List** to locate the segment to add.
- 5 Select a segment and click **Apply**.
- 6 Click **OK** on the **Add Segment to Message** dialog box.

Limitations in message variations

Limitations in message variations include:

- The standard C-Service and N-Service DIMSE Headers are included as segment files.
- The standard Data Element is included as a field.
Any Data Element that is tagged as retired in the 2014a spec is not included as a standard Data Element. If you require using a retired Data Element, then you must use the IDE to add retired fields to the DICOM variant. Otherwise, the retired Data Element cannot be displayed in the IDE and handled by the engine.
- Standard DICOM messages that are ready to use are not provided, because the DICOM message is not defined in the DICOM specification. You must define your own DICOM message in the DICOM variant according to the incoming DICOM message.

A DICOM message should be composed of only two segments in separate groups.

Example

```
{
C_STORE_RQ_DIMSE    <- Group 0 (Standard C STORE Request DIMSE)
}
{
C_STORE_DATASET     <- Group 1 (User defined C STORE DATASET)
}
```

Group 0 should be the DIMSE Header segment. You can use the standard C-Service or N-Service DIMSE Header segment file.

Group 1 should be the DATASET part of a DICOM message. You can use the IDE to configure the DATASET segment file or use `hcidicomdefgen` to create the DATASET segment file based on a DICOM message.

IOD (Information Object Definition)

The DICOM Information Model defines the structure and organization of information that are related to the communication of medical images.

An IOD (Information Object Definition) is an object-oriented abstract data model used to specify information about real-world objects. An IOD provides communicating application entities with a common view of the information to be exchanged.

An IOD does not represent a specific instance of a real-world object, but a class of real-world objects that share the same properties.

Cloverleaf provides a standard IOD as a root-level format definition.

iods folder

This folder contains the IOD definitions. For example:

```
Prologue
who:      18265
date:     Tue Mar 03 15:13:16 2015
version:  4.0
type:     iod
name:     RT Image IOD
ldlvers:  2014
end_prologue
{MODULE Patient} {REQD 0} {REPEAT 0} {REPCNT 0} {LEVEL 0}
{MODULE Clinical_Trial_Subject} {REQD 0} {REPEAT 0} {REPCNT 0} {LEVEL 0}
{MODULE General_Study} {REQD 0} {REPEAT 0} {REPCNT 0} {LEVEL 0}
{MODULE Patient_Study} {REQD 0} {REPEAT 0} {REPCNT 0} {LEVEL 0}
.....
```

modules folder

This folder contains Module definitions. For example:

```
Prologue
who:      18265
date:     Tue Mar 03 15:13:17 2015
version:  4.0
type:     mod
name:     Patient Module
ldlvers:  2014
end_prologue
{FIELD 0010,0010} {REQD 0} {REPEAT 0} {REPCNT 0} {LEVEL 0}
{FIELD 0010,0020} {REQD 0} {REPEAT 0} {REPCNT 0} {LEVEL 0}
{MACRO Issuer_of_Patient_ID} {REQD 0} {REPEAT 0} {REPCNT 0} {LEVEL 0}
{FIELD 0010,0030} {REQD 0} {REPEAT 0} {REPCNT 0} {LEVEL 0}
.....
```

macros folder

This folder contains Module definitions. For example:

```
Prologue
who:      18265
date:     Tue Mar 03 15:13:15 2015
version:  4.0
type:     mac
name:     Issuer of Patient ID Macro
```

```
ldlvers: 2014
end_prologue
{FIELD 0010,0021} {REQD 0} {REPEAT 0} {REPCNT 0} {LEVEL 0}
{FIELD 0010,0024} {REQD 0} {REPEAT 0} {REPCNT 0} {LEVEL 0}
{FIELD FFFE,E000} {REQD 0} {REPEAT 0} {REPCNT 0} {LEVEL 1}
{FIELD 0040,0032} {REQD 0} {REPEAT 0} {REPCNT 0} {LEVEL 2}
.....
```

Addressing

Cloverleaf supports using IOD,MODULE, and MACRO in the message address instead of each particular data element. For example:

```
1(0).C_STORE_MR_IOD(0).MR_Image(0).Image_Pixel(0).Image_Pixel(0).0028,0101(0)
```

When the exact position of the data element in an incoming message is unknown, then use an address without any fixed index before the data element. In this way, the engine searches the data element in the particular level for it.

DICOM and HL7

DICOM and HL7 messages can be transformed between each format.

This table shows the similarities between DICOM and HL7 messages:

	DICOM	HL7	Notes
1	Hierarchical & LDL	Hierarchical	
2	Commands	Trigger Events	
3	Command Group/DIMSE	SH Segment / Message Type	Message header. Sub-fields form TrxID
4	DICOM Information Object (IOD)	Message type specific segments	Data for the message
5	Service-Object Pairs SOP(= DIMSE + IOD)	Specific message, for example, ORU, ADT	Message header + data
6	Elements	Fields	Both have types
7	Elements types (called VR)	Field Types	
8	Request/Response possible	Request/Response possible	

This table shows the differences between DICOM and HL7 messages:

	DICOM	HL7	Notes
1	Upper level protocol/PDU	None	Items form TrxID also

	DICOM	HL7	Notes
2	Text and Binary	Text	Encoding depends on the field type
3	Elements are length delimited	Fields, components, sub-components, and segments are character delimited	
4	Elements can be grouped within a sequence (SQ element)	Grouping is well-defined. Messages contain segments, which contain fields, and so on.	

Additional information in message metadata

The engine saves other information in message metadata to help the DICOM protocol work.

Information that is saved to `DriverControl` includes:

- `CallingAET`: The modality sending the request.
- `CalledAET`: The modality expected to reply the request.
- `ReplyAET`: The modality replies the request. This is appended only to the reply message.
- `AbstractSyntax`: Used to construct the TRXID.
- `TransferSyntax`: Used to parse and encode the DICOM message.

Examples

Data message (DICOM request)

```
msgDriverControl : {CallingAET FINDSCU} {CalledAET DICOMTEST} {AbstractSyntax
1.2.840.10008.5.1.4.31}
{TransferSyntax 1.2.840.10008.1.2}
```

Reply message (DICOM response)

```
msgDriverControl : {CallingAET FINDSCU} {CalledAET DICOMTEST} {AbstractSyntax
1.2.840.10008.5.1.4.31}
{TransferSyntax 1.2.840.10008.1.2} {ReplyAET DICOMTEST}
```

Logging and debugging

Logging and debugging operations are performed through the dicm Engine Output module and Dcmtk logging.

dicm Engine Output module submodules:

- `init`: Initialization

- open: Open a file or make a connection
- read: Read data
- write: Write data
- close: Close a file or connection

For example:

```
[dicm:init:INFO/1: CL_SCP:09/05/2014 03:58:00] Initializing
[dicm:open:DEBUG/0: CL_SCP:09/05/2014 03:58:00] scp using local port:11112
[dicm:read:INFO/1: CL_SCP:09/05/2014 03:58:05] Reading
[dicm:wrt:INFO/1: CL_SCP:09/05/2014 03:58:07] Writing
[dicm:clse:INFO/1: CL_SCP:09/05/2014 03:58:16] Shutting down
```

Dcmtdk logging

This provides logging detail levels:

- disable
- fatal
- error
- warn
- info
- debug
- trace

The configuration is saved in NetConfig:

```
{ PROTOCOL {
  { DICOM trace }
...
}
```

Dcmtdk logging is useful in checking the raw data received/sent by Cloverleaf and debugging the entire DICOM message transferring process. For example:

```
D: setting network send timeout to 60 seconds
D: setting network receive timeout to 60 seconds
T: DUL FSM Table: State: 1 Event: 4
T: DUL Event: Transport connection indication
T: DUL Action: AE 5 Transport Connect Response
T: Read PDU HEAD TCP: 01 00 00 00 01 40
T: Read PDU HEAD TCP: type: 01, length: 320 (140)
T: DUL FSM Table: State: 2 Event: 5
T: DUL Event: A-ASSOCIATE-RQ PDU (on transport)
T: DUL Action: AE 6 Examine Associate Request
D: PDU Type: Associate Request, PDU Length: 320 + 6 bytes PDU header
D: 01 00 00 00 01 40 00 01 00 00 44 49 43 4f 4d 54
D: 45 53 54 20 20 20 20 20 20 20 53 54 4f 52 45 53
D: 43 55 20 20 20 20 20 20 20 20 00 00 00 00 00
D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
.....
```

This shows association information, including A-ASSOCIATE-RQ and A-ASSOCIATE-AC.

```
D: ===== BEGIN A-ASSOCIATE-RQ =====
D: Our Implementation Class UID: 1.2.276.0.7230010.3.0.3.6.1
D: Our Implementation Version Name: OFFIS_DCMTK_361
D: Their Implementation Class UID: 1.2.40.0.13.1.1
D: Their Implementation Version Name: dcm4che-3.3.3
```

```

D: Application Context Name: 1.2.840.10008.3.1.1.1
D: Calling Application Name: STORESCU
D: Called Application Name: DICOMTEST
D: Responding Application Name:
D: Our Max PDU Receive Size: 16384
D: Their Max PDU Receive Size: 16378
D: Presentation Contexts:
D:   Context ID: 1 (Proposed)
D:     Abstract Syntax: =VerificationSOPClass
D:     Proposed SCP/SCU Role: Default
D:     Proposed Transfer Syntax(es):
D:       =LittleEndianImplicit
D:   Context ID: 3 (Proposed)
D:     Abstract Syntax: =MRImageStorage
D:     Proposed SCP/SCU Role: Default
D:     Proposed Transfer Syntax(es):
D:       =LittleEndianImplicit
D:   Context ID: 5 (Proposed)
D:     Abstract Syntax: =MRImageStorage
D:     Proposed SCP/SCU Role: Default
D:     Proposed Transfer Syntax(es):
D:       =JPEG2000
D: Requested Extended Negotiation: none
D: Accepted Extended Negotiation: none
D: Requested User Identity Negotiation: none
D: User Identity Negotiation Response: none
D: ===== END A-ASSOCIATE-RQ =====
T: DUL FSM Table: State: 3 Event: 6
T: DUL Event: A-ASSOCIATE resp prim (accept)
T: DUL Action: AE 7 Send Associate AC
D: Constructing Associate AC PDU
D: pdDICOMDcmSCP: Association Acknowledged
I: Association Acknowledged (Max Send PDV: 16366)
D: ===== BEGIN A-ASSOCIATE-AC =====
D: Our Implementation Class UID: 1.2.276.0.7230010.3.0.3.6.1
D: Our Implementation Version Name: OFFIS_DCMTK_361
D: Their Implementation Class UID: 1.2.40.0.13.1.1
D: Their Implementation Version Name: dcm4che-3.3.3
D: Application Context Name: 1.2.840.10008.3.1.1.1
D: Calling Application Name: STORESCU
D: Called Application Name: DICOMTEST
D: Responding Application Name: DICOMTEST
D: Our Max PDU Receive Size: 16384
D: Their Max PDU Receive Size: 16378
D: Presentation Contexts:
D:   Context ID: 1 (Accepted)
D:     Abstract Syntax: =VerificationSOPClass
D:     Proposed SCP/SCU Role: Default
D:     Accepted SCP/SCU Role: Default
D:     Accepted Transfer Syntax: =LittleEndianImplicit
D:   Context ID: 3 (Accepted)
D:     Abstract Syntax: =MRImageStorage
D:     Proposed SCP/SCU Role: Default
D:     Accepted SCP/SCU Role: Default
D:     Accepted Transfer Syntax: =LittleEndianImplicit
D:   Context ID: 5 (Accepted)
D:     Abstract Syntax: =MRImageStorage
D:     Proposed SCP/SCU Role: Default
D:     Accepted SCP/SCU Role: Default
D:     Accepted Transfer Syntax: =JPEG2000
D: Requested Extended Negotiation: none
D: Accepted Extended Negotiation: none
D: Requested User Identity Negotiation: none
D: User Identity Negotiation Response: none
D: ===== END A-ASSOCIATE-AC =====

```

In addition to the hex raw data, it also shows human readable data.

```

T: # Dicom-Data-Set
T: # Used TransferSyntax: Little Endian Implicit
T: (0000,0002) UI =MRImageStorage # 26, 1 AffectedSOPClassUID
T: (0000,0100) US 1 # 2, 1 CommandField

```

```
T: (0000,0110) US 1 # 2, 1 MessageID
T: (0000,0700) US 0 # 2, 1 Priority
T: (0000,0800) US 0 # 2, 1 CommandDataSetType
T: (0000,1000) UI [1.3.12.2.1107.5.2.5.11090.5.0.5825067617589833] # 46, 1 AffectedSOPInstanceUID
```

Handling large messages

Cloverleaf retains all information in memory, and copies it during translation and routing, if required.

If the DICOM is too large, then:

- 1 In Network Configurator, open the **Process Configuration** dialog box (**Process > Configure**).
- 2 On the **Properties** tab, select **Disk-based Queueing** and set a larger **Threshold in Megabytes**. By doing so, the engine saves the message to disk but still provides transparent operation to users.

DICOM definition generate tool

Use `hcidicomdefgen` to generate a DICOM dataset segment definition file according to an incoming message.

```
hcidicomdefgen -V version -v variant -s service_name -c transfersyntax input_message
```

- `-V version` is the DICOM version.
- `-v variant` is the DICOM variant.
- `-s service_name` is the DICOM service name.
For example: `C_STORE_RQ`, `C_FIND_RSP`,...
- `-c transfer_syntax` is the transfer syntax of the DICOM message.
- `input_message` is the input DICOM message.

This command:

- Creates a variant under `$HCISITEDIR/formats/ldl/version/`.
- Creates a segment named `<service_name>_dataset`, and generates the content according to the input message parsing result.
- Creates `service_name_MSG` under `$HCISITEDIR/formats/ldl/version/variant/messages`, which contains:
 - `service_name_DIMSE`: This should already be defined under `$HCIR00T/formats/ldl/<version>/<variant>/segments`. This command does not generate the definition file for the DIMSE header part.
 - `service_name_dataset`: Content is generated based on the dataset part of the input DICOM message. This file is located under `$HCISITEDIR/formats/ldl/<version>/<variant>/segments`.

Lookup Table Configurator

Use the Lookup Table Configurator to configure the lookup tables used to convert message data. Lookup tables perform code translations within one or more translation configurations.

A lookup table is called up within a translation configuration using the Translation Configurator. It is used in the Table action for a specific field.

For example, you can configure a lookup table to convert numeric codes to alpha codes and back again.

Before using the Lookup Table Configurator, verify:

- The types of transactions that are processed by each connection.
- The record layouts for those transactions.
- The types of data that are handled by each transaction.
- The way that data is processed or translated.

"tbl" extension

All tables must have the .tbl extension to be viewed in the Site Manager and lists within the Translation Configurator.

If these tables are created from within the dialog box, then they already have the correct .tbl extension.

Sites that are brought forward from previous version that do not use the .tbl extension must have the tables files renamed to include the .tbl extension. They must also have the references updated in the translation that uses the table.

Table lookup files must end in .tbl when selecting lookup table files in Translation Configurator and Testing Tool.

The Translation Configurator only lists *.tbl entries and the Testing Tool give an error if this is not the case.

Open Lookup Table

When editing a TABLE action in an xlate, you can select **Open Lookup Table** to open the lookup table.

An **Open Lookup Table** menu option is also on the menu of the Action TABLE and **Edit** menu. There is also one in the TABLE action properties panel, after the **Table** list.

This option is disabled when the **Table** is empty.

Lookup table definition: Table lookup

If it is a new table, then select **Edit > New Item > Append**.

Specify the input and output data values in the In and Out cells.

Use **Edit > Trim Cells** to remove extra spaces on table entries. Otherwise, the lookup table file would contain unnecessary spaces that could affect the behavior of the tbllookup command. This option is also available on the context menu when right-clicking a table entry.

Edit menu

- **Trim Cells**

Removes extra spaces on Lookup Table entries.

Table Control Values pane

Click **Bi-directional** to reconfigure the dialog box to reveal the side information for each pair.

If the table lookup fails, then selecting **Use Original Value** keeps the original value of the field being loaded from the table lookup.

Select **Use Default Value** to use what is defined in the field. Specify a default string to use for data conversion if an input data value is not in the lookup table.

For security, select **Set Default Value as Masked Field**.

See [Table security](#).

Bi-directional operation

Bi-directional specifies that the table's data pairs translate in both directions (Side 1 to Side 2 or Side 2 to Side 1).

Data input values (or left-side input) must be unique. If using bi-directional pairs, then unique data output values, or right side, are required, as both sides are used for input.

Side 1 assigns a name to the entries on the left side of the table. Use this name to reference these values when configuring a table lookup with Translation Configurator. The default name for this side of the table is Input.

Side 2 assigns a name to the entries on the right side of the table. Use this name to reference these values when configuring a table lookup with Translation Configurator. The default name for this side of the table is Output.

Lookup table definition: Database lookup

Lookup is supported in the translation TABLE action and as a `dblookup` Tcl extension. It uses the Database Schema Configurator to configure database connections. It also uses the Lookup Table Configurator to configure the Database Lookup SQL statement and IN/OUT column list.

The Database Lookup feature uses the xlate engine to look up the matched values through the database.

New configurations are saved as a `.tbl` file.

The Translation Configurator applies the `.tbl` file in a Table action.

During runtime, the translation engine applies the source field values to the query statement that is defined in the `.tbl` file. It uses this to look up the specified output value.

Database Lookup action in translation

The **Database Lookup** options in the Lookup Table Configurator use the TABLE action to do the lookup. The tables use column name and value address mapping. Multiple sources to multiple destinations are supported.

Information storage

The `DBLookup.jar` file is located in the `$HCIRoot/lib/java` folder.

Editing database statements

If required, then you can make text edits to database statements. Select the type of database statement by clicking **SQL** or **Stored Procedure**. Both of these open the **Stored Procedure Out Parameters** dialog box.

This dialog box has these checkboxes where you can define the expected return from the stored procedure:

- **Return Code**
- **Out Parameters**

One of these must be selected. All are selected by default.

Result Set is optional. Sometimes, a result set is returned by a stored procedure. To use this in **Out Parameters**, specify the column names separated by a comma. Otherwise, if the result set is not required, you can leave the field blank.

Database Lookup options

To maintain compatibility with previous versions, **Table Lookup** reconfigures the GUI to the pre-6.2 configuration. This is the default.

The **Basic Database Lookup** option is suitable for users who require to look up data from the In field in a database table. It is also used to view and return the matched value in the Out field.

Selecting **Advanced Database Lookup** is for users who require to do a more complex query in the database.

Shared options are:

- **Bi-directional**
This is unavailable, as Database Lookup only supports single-direction.
- **Use Original Value**
If the lookup fails, this options keeps the original value of the field being loaded from the lookup. This option is unavailable if you have defined multiple In columns.
- **Use Default Value**
Select this to use what is defined in the field. Specify a default string to use for data conversion if an input data value is not in the lookup table.
- **Set Default Value as Masked Field**
Sets the default field as a masked field.

See [Table security](#).

Configuring a basic database lookup

- 1 Open the **Database Connection** menu and select a database. The list includes all configured database connections, including master (primary) site connections. This is required.
- 2 Click the **Table** button and make a selection.

This opens a **Table Selection** dialog box that lists all the tables/views found in the selected Database Connection. Only one table can be selected at a time. This is required.

- 3 Select the In and Out columns. These are required.

Clicking the button opens the **Column Selection** dialog box that lists all the columns in the selected table/view. Multiple selections are permitted. Selected values populate the field as a comma separated list.

The **SQL Statement** field shows the generated SQL statement and is read-only. This statement is automatically generated from Table, In, and Out.

The generated SQL statement is similar to:

```
select
  columnOut1, columnOut2
from
  table1
where
  columnIn1=<columnIn1>
  and
  columnIn2=<columnIn2>
```

columnIn1 and columnIn2 are placeholders that represent the configured In columns. During runtime, the translation engine replaces placeholders with their corresponding real values.

Configuring an advanced SQL database lookup

- 1 Select **Advanced Database Lookup**.
- 2 Open the **Database Connection** menu and select a database. The list includes all configured database connections, including master (primary) site connections. This is required.
- 3 For Type, select **SQL**.
- 4 In the **SQL Statement** field, you can directly specify any valid query statement. You can also click **Configure** to open the **Inbound (Outbound) SQL Statement Configurator** dialog box where you can compose the query statement. This is required.

This dialog box provides functions to help you compose the SQL statement.

In the **Inbound (Outbound) SQL Statement Configurator** dialog box, all real condition values must be replaced with the appropriate placeholder. SQL content can span multiple lines and must be a query statement. Otherwise, the GUI prompts an error message.

- 5 Select the In and Out columns. These are required.

In/Out show the names of specified table/view columns. Clicking the button opens the **Column Selection** dialog box that lists all the columns in the selected table/view. Multiple selections are permitted. Selected values populate the field as a comma separated list.

The **Out** field shows all the columns that are the output of the Translation Table action.

Configuring an advanced stored procedure database lookup

Note: If there are multiple stored procedures defined in the DB Lookup table stored procedure, then only the first stored procedure is run in the engine. This applies even though you can add multiple stored procedures through the GUI.

To configure an advanced stored procedure database lookup:

- 1 Select **Advanced Database Lookup**.
- 2 Open the **Database Connection** menu and select a database. The list includes all configured database connections, including master (primary) site connections. This is required.
- 3 For Type, select **Stored Procedure**.
- 4 In the **Stored Procedures** field, you can directly specify any valid query statement. You can also click **Configure** to open the **Stored Procedure Configurator** dialog box where you can compose the query statement. This is required.

This dialog box provides functions to help you compose the stored procedure.

In the **Stored Procedure Configurator** dialog box, all real condition values must be replaced with the appropriate placeholder. Stored procedure content can span multiple lines and must be a query statement. Otherwise, the GUI prompts an error message.

- 5 Select the appropriate Procedure, Input Parameters, and Output Parameters.
The ORACLE database supports defining a result set using Output Parameters.
Prefix codes for Output Parameters are:
 - `_CLRC_`: Return code.
 - *Parameter name* and "OUT": Out parameter.
 - `RS_` and the *Parameter name*: Manually specify this in the text area.
- 6 If you select **Return code**, then a `?:` is placed at the beginning of the SQL statement. The `_CLRC_` variable can receive a value from runtime.
If required, then you can select a parameter to modify from the Input or Output list.
- 7 When finished, select **OK**.

Example advanced SQL statement configuration

To define these SQL statement as an Advanced Database Lookup SQL:

```
select * from recorder_table where patientid = '001' and date = '2013-7-1' and doctorid = '003'
```

Placeholder fields can be used similar to these examples:

- SQL Statement:

```
select * from patient_table where patientid =patientid and date=date and doctorid=doctorid
```

- In:

```
patientid,date,doctorid
```

- Out:

```
patientid,date,doctorid,patientname,doctorname,result
```

Configuration requirements

The related configurations that are used in Database Lookup are:

Configuration	Location
JDBC drivers	By default, these are installed under \$HCIR00T/clgui/lib.
JDBC driver configuration	\$HCIR00T/server/database.ini
Database connection configuration	\$HCISITEDIR/dbconfiguration.ini
Database Lookup Table configuration	\$HCISITEDIR/.tbl

Error handling of database lookup

Database Lookup returns an error as long as there is an error during the query.

In all cases, meaningful error messages are logged in the engine log file. Meaningful, human, and machine readable errors are returned to the user for action by UPoC scripts or other means.

The table control values that are defined in the .tbl file are used when the query is run and no result set is returned.

In the database lookup TABLE action, you can select **Skip** to ignore the error, or **Error** to move the message into the error database.

Error handling of the hcidblookup Tcl extension

If there is an error during running hcidblookup, then the error information is appended to the Tcl result. You can use `catch {dblookup table value} errMsg` to get the error information (`$errMsg`) of hcidblookup.

If there are no errors during running hcidblookup but no data is queried, then the default value is set to the Tcl result.

Variables for database lookup

Two variables are available in database lookup. These can be used after running the database lookup table action to indicate the detail error information.

- `@tbl_errno`
This integer variable is for the TABLE action to indicate the query result. Before running the TABLE action, `@tbl_errno` is initialized to 0. After running the action:
 - `@tbl_errno` is set to -1 if there is an error during query.
 - `@tbl_errno` is set to 1 if there are multiple rows returned.

You can use `@tbl_errno` in subsequent actions to do error handling. The error number and error information of database lookup are also dumped in the engine error log.

- If no data is returned or there are errors returned from the TABLE Action, then the output field is set to the default value.
- If multiple rows are returned from the TABLE Action, then `@tbl_errno` is set to 1. A Tcl list of the field value for all rows is set to the output field, and a warning is written to the engine log.
- `@tbl_errdetail`
This variable indicates the detail error information.

Tcl variables

The Tcl variables `$xlateTblErrNo` and `$xlateTblErrDetail` are added to the Database Lookup TABLE post proc fragment. You can use these directly in the Tcl UPoC.

- `$xlateTblErrNo` has the same meaning as `@tbl_errno`.
- `$xlateTblErrDetail` has the same meaning as `@tbl_errdetail`.

Multiple rows returned from the database lookup TABLE action

Multiple return values can be handled from a database run in the database lookup TABLE action. The engine handles multiple values returned in a Tcl list and notifies that multiple values were returned.

Generally, there should be only one row returned in the database lookup TABLE action. If there are multiple rows returned, then the expected field of each row is combined to a list and set to the specific destination's out column.

For example:

```
SELECT intFLD,dateFLD,flag FROM Table_sanity WHERE ID=<ID> and strFLD=<strFLD> ORDER BY ID
```

Rows returned:

```
test1, 2014-04-01, 0
test2, 2014-04-02, 0
test3, 2014-04-03, 0
```

Destination out columns:

```
intFLD is set to {test1 test2 test3}
dateFLD is set to {2014-04-01 2014-04-02 2014-04-03}
flag is set to {0 0 0}
```

Database lookup Tcl extension

dblookup

```
dblookup ?-maxrow ?count?? ?-metacolumnname? table value ?value...?
```

This looks up database data through a database lookup table that is based on the given value. The table is a database lookup file name; the file extension can be omitted.

The input values provided by `dblookup` are set as the IN column values one-by-one. The input value account equals the IN column account that is defined in the `.tbl` file. If they are not matched, then an error is prompted and the database lookup terminated.

`-maxrow ?count?` sets the maximum returned row count. By default, at most one record is returned from the `dblookup` command. With this option, at most the specific `count` records are returned. To get all the selected records, set this option without `count`.

`-metacolumnname` gets the column names from result set metadata. With this option, both the selected database data and column names of result set metadata are returned in a keyed list.

dblookupdestroy

This destroys the related resources, for example, JNI context and database connection, used by `dblookup`. This command returns an empty string.

```
dblookupdestroy table
```

Using multi-byte encoding in the Lookup Table

For specific information, see [Multi-Byte encoding](#).

HIPAA external code sets

External code sets are tables of acceptable code values and their associated descriptive information that is used in HIPAA standard transactions. They are referred to as external because they are defined by organizations other than the ones that define the transaction formats.

Several HIPAA code sets in lookup table format are included with the system. Because these lookup tables are large and non-site-specific, they are kept in the `$HCIR00T/Tables` directory.

Previous versions had no product-supplied lookup tables. User-created lookup tables are kept in `$HCISITEDIR/Tables`.

The translation thread and the `hcitbllookup` Tcl extension search for lookup tables in `$HCISITEDIR/Tables` first and then `$HCIR00T/Tables`. That is, the site `Tables` directory is searched first, and if the table is not found, the root `Tables` directory is searched.

Modifying a table

Lookup tables are viewed and modified with the Lookup Table Configurator. The Lookup Table Configurator searches only in the user-created lookup tables directory (`$HCISITEDIR/Tables`).

To view or modify a HIPAA external code set lookup table, copy or link the table to the `Tables` directory in the site. Do this before running the Lookup Table Configurator. The modifications are made to the table in the site and are used in processing according to the search order that is mentioned above.

Code set Tcl procedure template

A HIPAA code set Tcl procedure template is provided that gives an example of querying a lookup table for a particular key.

Sorting and searching table items

To sort items in the Lookup Table Configurator, click the column headers. Each click switches the sorting orientation (ascending/descending). Sorting criteria is case-sensitive.

For items with the same value in the same column, the sorting function groups the items together. The value of the other column is used as the secondary sort.

Items that are highlighted before being sorted retain their highlighting after sorting.

After sorting, an asterisk (*) displays in the title bar, meaning that the configuration is unsaved. Saving the configuration also saves the sort order.

On the toolbar is a text box and **Search**. This is the binoculars icon.

Specify characters in the text box and click **Search** to begin a search. Searching criteria are case-insensitive and substring match. If a match is found, then it is highlighted. The grid is searched horizontally first, then vertically.

Table security

Use **File > Set Password** to set a new password to an unprotected file or to modify the password of a protected file.

With a protected file, when you select **File > Set Password**, the **Modify Password** dialog box opens, where you can change the existing password.

With a non-protected file, when you select **File > Set Password**, the **Password Setting** dialog box opens, where you can specify a new password.

The password settings are saved in the table file, which is similar to:

```
# Translation lookup table
#
prologue
  who: dwc
  date: June 18, 2013 3:03:17 PM CDT
  outname: output
  inname: input
  bidir: 0
  type: tbl
  version: 4.0
end_prologue
#
password=cGFzc3dvcmQ=
dflt_passthrough=0
dflt=
dflt_encoded=false
#
TRIPS
PRIC1
encoded=0,0
#
TRIPM
PRIC2
encoded=0,0
#
MAIL
PRIC4
encoded=0,0
#
MISCS
TEST
encoded=0,0
#
HMISC
TEST1
encoded=0,0
#
CMISC
TEST2
encoded=0,0
#
UMISC
TEST3
encoded=0,0
#
OHMIS
TEST4
encoded=0,0
#
BMISC
TEST5
encoded=0,0
```

The value for `password` indicates the current file is password protected; this key does not exist if the file is not password protected.

The value for `dflt_encoded` indicates whether the default value is encoded.

- `true` means an encoded default value.
- `false` means that the specified default value is not encoded.

This key is missing if no default value is encoded.

The values for `encoded` specify whether the cells in each row are encoded. A value of 1 means an encoded cell. A value of 0 means that the cell is not encoded. This key does not exist if no cell in that row is encoded.

Removing the password

Select **File > Remove Password** to remove the password to a protected file. This opens a confirmation dialog box. Click **Yes** to continue.

Setting a masked field

The Lookup Table Configurator displays all cell values as clear text. To provide more security with protected data, use a masked field.

The masked field is used to provide encryption ability. You can set any cell as a masked field or reset it as a clear text cell.

The **Set as Masked Field** setting is protected by a file-level password. You must create or specify a valid password before changing any Masked Field setting.

For users who have a valid password, the masked field displays the values as clear text in blue color.

For users who do not have the password, it displays the value as asterisks.

- 1 Select a table cell and right-click to open a menu where you can select **Set as Masked Field**. Or, you can select a cell and then select **Edit > Set as Masked Field** from the menu bar.

When you change a masked field back to clear text, the menu option changes to Set as Plain Field.

A cell must be selected to enable this option.

- 2 If a password is not set and there is no masked setting for the current table, then the **Password Setting** dialog box opens. In this dialog box, you can create a password for the file.

If the password is correct, then all data in the table can be seen as clear text, and the masked fields text is blue.

If the password is not correct, then a `Password Invalid` message opens.

- 3 Click **OK** and then click **Cancel** on the **Enter Password** dialog box. A message opens stating that all masked fields are obscured.

The file opens when you click **OK**, but the text in the masked field is obscured, displayed as asterisks.

The **Set as Masked Field/Set as Plain Field** menu option is still enabled, although you must specify the correct password to use this option.

You can edit the value of both masked and non-masked fields, even if the password is invalid. The password is only used to protect the masked field setting.

After the password is entered successfully, the selected cell is set as a masked field and its text color is blue. As long as the file is not closed, later actions to set/reset the masked field do not prompt for a password.

If a password is not set, then you cannot change the cell property to a masked field. The cell keeps its original text color.

Note: To change a masked field back to a plain field, right-click the masked field and select **Set as Plain Field** from the list. The selected cell is reset as a plain field, which shows the value in its original color.

Encryption setting for default value

Select **Set Default Value as Masked Field** to set the default field as a masked field.

If this is the first time to set a masked field, then the **Password Setting** dialog box opens for you to create a password for the file.

After the password is set, the specified default value is set to a masked field and its text color is blue. Later actions to set or reset a masked field do not prompt for the password again as long as the file remains open.

If the password is not set, then you cannot change the default field property to a masked field.

Bi-directional usage

When **Bi-directional** is selected, two **Set Default Value as Masked Field** check boxes are added beneath each side. The default behavior of the **Use Default Value** field on the left side is the same as the original single one.

When you select **Set Default Value as Masked Field**, its respective default text field is set as a masked field. This displays in blue.

When you clear **Set Default Value as Masked Field**, its respective default text field is set as a plain field. This displays in the original color.

Reserved table names

These names are reserved and should not be used:

adaLong.tbl	adaShort.tbl	canProvinces.tbl
carcDesc.tbl	carcNote.tbl	countries.tbl
cptLong.tbl	cptMedium.tbl	cptModifiers.tbl
cptShort.tbl	currencies.tbl	currenciesNum.tbl
hccsccDesc.tbl	hccsccNote.tbl	hcpcsActEffDate.tbl
hcpcsActionCode.tbl	hcpcsAnesBUQ.tbl	hcpcsAPayGrpCode.tbl
hcpcsAPayGrpEffDate.tbl	hcpcsBETSC.tbl	hcpcsCodeAddDate.tbl
hcpcsCovCode.tbl	hcpcsCovIssues.tbl	hcpcsCrossRefCode.tbl
hcpcsLabCertCode.tbl	hcpcsMediCarMRS.tbl	hcpcsMEffDate.tbl
hcpcsModLong.tbl	hcpcsModShort.tbl	hcpcsMPayGrpCode.tbl

hcpcsMPayPolInd.tbl	hcpcsMultPriceInd.tbl	hcpcsPriceInd.tbl
hcpcsProcLong.tbl	hcpcsProcShort.tbl	hcpcsProNoteNum.tbl
hcpcsStatuteNum.tbl	hcpcsTermDate.tbl	hcpcsTSC.tbl
hiecDesc.tbl	hiecSummDesc.tbl	icdLong.tbl
icdMedium.tbl	icdShort.tbl	naicALAL_cc.tbl
naicALAL_cn.tbl	naicALNM_cc.tbl	naicALNM_cn.tbl
naicCOMB_cn.tbl	naicCOMB_cs.tbl	naicCOMB_gc.tbl
naicCompanyStatus.tbl	naicCONM_cn.tbl	naicCONM_cs.tbl
naicCONM_fn.tbl	naicCONM_gc.tbl	naicCONM_sd.tbl
naicFRAT_a.tbl	naicFRAT_c.tbl	naicFRAT_cn.tbl
naicFRAT_cs.tbl	naicFRAT_fn.tbl	naicFRAT_gc.tbl
naicFRAT_p.tbl	naicFRAT_s.tbl	naicFRAT_sd.tbl
naicFRAT_z.tbl	naicGPAL_gn.tbl	naicGPNM_cn.tbl
naicGPNM_cs.tbl	naicGPNM_fn.tbl	naicGPNM_gc.tbl
naicGPNM_gn.tbl	naicGPNM_sd.tbl	naicHMDI_a.tbl
naicHMDI_c.tbl	naicHMDI_cn.tbl	naicHMDI_cs.tbl
naicHMDI_fn.tbl	naicHMDI_gc.tbl	naicHMDI_p.tbl
naicHMDI_s.tbl	naicHMDI_sd.tbl	naicHMDI_z.tbl
naicHMO_a.tbl	naicHMO_c.tbl	naicHMO_cn.tbl
naicHMO_cs.tbl	naicHMO_fn.tbl	naicHMO_gc.tbl
naicHMO_p.tbl	naicHMO_s.tbl	naicHMO_sd.tbl
naicHMO_z.tbl	naicLHSO_a.tbl	naicLHSO_c.tbl
naicLHSO_cn.tbl	naicLHSO_cs.tbl	naicLHSO_fn.tbl
naicLHSO_gc.tbl	naicLHSO_p.tbl	naicLHSO_s.tbl
naicLHSO_sd.tbl	naicLHSO_z.tbl	naicLIFE_a.tbl
naicLIFE_c.tbl	naicLIFE_cn.tbl	naicLIFE_cs.tbl
naicLIFE_fn.tbl	naicLIFE_gc.tbl	naicLIFE_p.tbl
naicLIFE_s.tbl	naicLIFE_sd.tbl	naicLIFE_z.tbl
naicPLAL_pae.tbl	naicPLAL_sd.tbl	naicPLNM_pae.tbl
naicPLNM_sd.tbl	naicPLNM_t.tbl	naicPROP_a.tbl

naicPROP_c.tbl	naicPROP_cn.tbl	naicPROP_cs.tbl
naicPROP_fn.tbl	naicPROP_gc.tbl	naicPROP_p.tbl
naicPROP_s.tbl	naicPROP_sd.tbl	naicPROP_z.tbl
naicTILE_a.tbl	naicTILE_c.tbl	naicTILE_cn.tbl
naicTILE_cs.tbl	naicTILE_fn.tbl	naicTILE_gc.tbl
naicTILE_p.tbl	naicTILE_s.tbl	naicTILE_sd.tbl
naicTILE_z.tbl	ndcApplicationNumber.tbl	ndcAppProdNumber.tbl
ndcbfDosage.tbl	ndcbfPackageType.tb	ndcbfRoutes.tbl
ndcbfUnits.tbl	ndcDosageForm.tbl	ndcDrugClass.tbl
ndcDrugClassNum.tbl	ndcFirmLabelCode.tbl	ndcFirmName.tbl
ndcFirmSeqNo.tbl	ndcFormulatIngName.tbl	ndcFormulatStrength.tb
ndcFormulatUnit.tbl	ndcLabelCode.tbl	ndcPackageCode.tbl
ndcPackageSize.tbl	ndcPackageType.tbl	ndcProdCode.tb
ndcRouteCodes.tbl	ndcRouteNames.tbl	ndcRxOtc.tbl
ndcStrength.tbl	ndcTblDClass.tbl	ndcTblDosageLong.tbl
ndcTblDosageShort.tbl	ndcTblRoute.tbl	ndcTblUnit.tbl
ndcTradeName.tbl	ndcUnit.tbl	protaxDescriptions.tbl
protaxTitles.tbl	states.tbl	

Lookup Table configuration example

This example shows how to configure a data lookup table. You enter the conversions in the table and save to a file. Then, you can use the table in a record translation specification to convert the input data to the output data.

- 1 Select **File > Set Password**, and specify a password.
- 2 Click **New Append**.
- 3 In the Table Entry pane, specify the first input data value in the In field.
- 4 Press **Tab** to go to the next field.
- 5 Specify the corresponding output data value in the Out field.
- 6 Press **Tab** to begin a new line.
- 7 Repeat steps 5-6 until all input and output data values are entered.

You can also use the arrow keys to move among the fields.

To add fields, use **Tab** or **New After/New Before**.

- 8 In the Table Control Values pane, specify **No Action** as the default value and select **Set Default Values** as Masked Field.
- 9 Click **Save**. This opens a file selection dialog box in which to save the table.
- 10 Specify a file name and save.

NCPDP Formulary and Benefit Configurator

Message format support of NCPDP SCRIPT and NCPDP Formulary and Benefit supports e-Prescribe.

The NCPDP Formulary and Benefit standard defines a file format by which a payer publishes formulary information to a consumer. Consumers are entities such as providers or clearinghouses. The NCPDP Formulary and Benefit standard is batch-oriented, being essentially a data push from the publisher to the consumer. There is no interactive use case. A small batch message flows back from the consumer to the publisher. This indicates the correctness of the file and any errors that may have happened.

Note: NCPDP Formulary and Benefit requires a license to operate. It is enabled with one of two possible license keys. An individual `cl-mm-ncpdpfab` license key enables NCPDP Formulary and Benefits. An overall `cl-mm-ncpdpall` license key enables all of the NCPDP formats (Telecom, Formulary and Benefit, and SCRIPT).

Copying and pasting across variants

Data elements, data segments, and messages can be copied and pasted across variants. When one of these is highlighted, right-clicking opens a menu with **Copy**, **Paste**, and **Delete** options.

Multiple selections are permitted.

Note: The **Paste** action does not apply between different HMD formats.

Batch types

The batch type can be selected for inbound and outbound EDI batch type.

Inbound batch

As the inbound Formulary and Benefit batch has no parameters, **Edit** is unavailable. Inbound batch processing splits the batch into individual transactions. It also attaches the field values from the batch header and trailer segments to each transaction message as driver control metadata.

- Driver Control is populated with a two-level keyed list.
- Inbound batch values are stored under the `INBOUND_BATCH` key.
- The batch header segment for a Formulary Load File is HDR and the trailer segment is TRL.
- The batch header segment for a response file is SHD and the trailer is STR.

- The entire batch message errors if the starting and ending segments are not valid. Special characters similar to "{", "}" and "\" in the data must be preceded by a backslash. This avoids corrupting the keyed-list structure.

This table shows the sub-keys that are populated by inbound batch processing:

Sub-key	Description
RECORD_TYPE	Header segment (HDR or SHD)
VERSION_NUMBER	
SENDER_ID	Only populated for a load file (HDR)
	PASSWORD
RECEIVER_ID	
SOURCE_NAME	Only populated for a load file (HDR)
	CONTROL_NUMBER
TRANS_DATE	
TRANS_TIME	
TRANS_TYPE	
ORIG_CONTROL_NUMBER	Only populated for a response file (SHD)
ORIG_TRANS_DATE	Only populated for a response file (SHD)
ORIG_TRANS_TIME	Only populated for a response file (SHD)
ACTION	Only populated for a load file (HDR)
EXTRACT_DATE	Only populated for a load file (HDR)
	FILE_TYPE
LOAD_STATUS	Only populated for a response file (SHD)
TOTAL_RECORDS	Only populated for a load file, in TRL at end
ROWS_IN_ERROR	Only populated for a response file, in STR at end
TOTAL_ERRORS	Only populated for a response file, in STR at end
BATCH_START_END	Batch start/end is used to indicate the first and last messages in a batch. This is necessary because the response batch must correspond to the inbound batch.

The first message that is extracted from an inbound batch has { {BATCH_START_END {S} } }.

The last message has { BATCH_START_END {E} }.

Intermediate messages have { BATCH_START_END { } }.

This constructs a response file that matches the inbound batch file.

For example, if a batch message has header and trailer as shown here, each extracted message has these driver control metadata. The `BATCH_START_END` varies.

Message data:

```
HDR|10|sndr|passwd|rcvr|source|1001|20060103|08150198|FRM|U|20051230|T
... formulary data load transaction segments ...
TRL|223
```

Drivercontrol:

```
{INBOUND_BATCH {{RECORD_TYPE {HDR}}} {VERSION_NUMBER {10}}} {SENDER_ID {sndr}} {PASSWORD {passwd}}
{RECEIVER_ID {rcvr}} {SOURCE_NAME {source}} {CONTROL_NUMBER {1001}} {TRANS_DATE {20060103}}
{TRANS_TIME {08150198}}
{TRANS_TYPE {FRM}} {ACTION {U}} {EXTRACT_DATE {20051230}} {FILE_TYPE {T}} {LOAD_STATUS {}} {TO
TAL_RECORDS {223}}
{ROWS_IN_ERROR {}} {TOTAL_ERRORS {}} {BATCH_START_END {S}} {}}
```

Outbound batch

The outbound batch has an edit screen with these parameters. These are configured from the **Outbound** tab of the Network Configurator's **Thread Configuration** dialog box.

Note: The only selection for version is "1.0." The **Message Count** and **Interval** fields work the same as with other EDI batches. All of the header fields are required in the final message. They can be left blank on the screen as they can be set and overridden with DRIVERCONTROL in each message.

This table shows the options on the **Outbound** tab of the Network Configurator's **Thread Configuration** dialog box:

Option	Description
Transmission File Type	Valid selections are FRM - Formulary and Benefit Load and FRE - Formulary and Benefit Response . This is stored in NetConfig as FRM or FRE.
Sender ID	Alphanumeric text up to 30 characters.
Sender Password	Alphanumeric text up to 10 characters. This displays as asterisks and is stored encrypted. When this screen displays, this field displays as 10 asterisks unless it is blank, in which case it displays as blank.
Receiver ID	Alphanumeric text up to 30 characters.
Source Name	Alphanumeric text up to 35 characters
Action	Valid selections are Full Replace and Update . This is be stored in NetConfig as F or U.
Extract date	Valid selections are Today's date and No default . This is be stored in NetConfig as T or {}.

Option	Description
File Type	Valid selections are Production and Test . This is stored in NetConfig as P or T.

Control Number Counter

This is the counter name used to generate control numbers. A standard counter file with this name is created in the process directory if one does not already exist. The created counter starts at 1 and wraps after 999999999. This field can be left blank in which case no counter is used and the user must supply the control number in DRIVERCONTROL. The counter field is encoded to be from 1 to 9 characters in length with no padding or leading zeros.

DRIVERCTL values

Similar to the other batch types, these values can be supplied or overridden with settings in DRIVERCTL metadata. There are also some values not on the screen that can be set in DRIVERCONTROL. These settings are sub-keys under the OUTBOUND_BATCH key.

Because there are multiple messages coming in, each with potentially different driver control values, the driver control values accumulate. The DRIVERCTL fields accumulate from each message, but blank DRIVERCTL do not erase.

For example, if message A has driver control fields:

```
ABC: haha
CDF: hoho
```

And messages B has driver control fields:

```
CDF: hehe
```

Then the resulting DRIVERCTL when the batch is created is:

```
ABC: haha
CDF: hehe
```

Typically, the DRIVERCTL values for each message should be the same. This method attempts to be as robust as possible.

Sub-keys under OUTBOUND_BATCH

This table shows the sub-keys under OUTBOUND_BATCH used by outbound batch processing:

Sub-key	Description
RECORD_TYPE*	If not populated by driver control, then this is set to HDR when TRANS_TYPE is FRM or set to SHD when TRANS_TYPE is FRE. Otherwise, this accepts the value that was passed by driver control.
VERSION_NUMBER	

Sub-key	Description
SENDER_ID	Only written for load file (HDR).
SENDER_PASSWORD	
RECEIVER_ID	
SOURCE_NAME	SOURCE_NAME is only written for load file (HDR).
COUNTER	COUNTER is the counter file name to generate the transaction control number field. This is ignored if CONTROL_NUMBER is specified.
CONTROL_NUMBER*	CONTROL_NUMBER is for when a control number is set with driver control. This is remembered and used for every outbound batch until it is updated by another value. If an outbound control counter file is specified and a CONTROL_NUMBER sub-key is supplied, then the sub-key value is used and the counter does not increment.
TRANS_DATE*	If not populated by driver control, then this is set to the current date when the batch is processed.
TRANS_TIME*	If this is not populated by driver control, then this is set to the current time when the batch is processed. The time stamp specifies hundredths of a second to be displayed. Note that this TCL version only supports seconds, so the hundredths portion is always "00".
TRANS_TYPE	This is FRM or FRE.
ACTION	This is F or U.
EXTRACT_DATE	This is T or { }. Driver control can override this with the correct time stamp. If the value is still T when the batch is processed, then the current date is substituted. This is only written for a load file (HDR).
FILE_TYPE	FILE_TYPE is T or P.
LOAD_STATUS*	LOAD_STATUS is only meaningful for a response batch and is written to SHD. The default is 01 for "file loaded correctly". A failed message sets this field to the appropriate error value for the entire batch.

Sub-key	Description
TOTAL_RECORDS*	TOTAL_RECORDS is only meaningful for a request batch and is written to HDR. This is calculated from the number of messages passed in that contained data and cannot be overridden.
ROWS_IN_ERROR*	
TOTAL_ERRORS*	
	ROWS_IN_ERROR and TOTAL_ERRORS are only meaningful for a response batch and are written to STR. These are calculated from the number of SDT messages and cannot be overridden.
BATCHSEND*	This is a special key that forces the creation of a batch ending with the current message. This is identical to the X12 and NCPDP batch functionality. Any string in this key causes the message to send. Blank, or no key, does not.

Note: The asterisk (*) indicates the sub-key is not available on the configuration screen.

The total records field in the load file trailer cannot be overridden because it is calculated from the number of messages in the batch.

Outbound batch processing also uses some sub-keys from the INBOUND_BATCH key of DRIVERCONTROL. These are used to populate the originating fields in a response batch header. Using the INBOUND_BATCH values for these keys makes the originating numbers automatically populate if the inbound DRIVERCONTROL is copied to the outbound messages.

Sub-keys under INBOUND_BATCH

This table shows the sub-keys used by outbound batch processing:

Sub-key	Description
CONTROL_NUMBER	This populates "Transmission Number, Originating" in the header. This applies only to a response type batch TRANS_TYPE=FRE.
TRANS_DATE	This populates "Transmission Date, Originating" in the header. This applies only to a response type batch TRANS_TYPE=FRE.
TRANS_TIME	This populates "Transmission Time, Originating" in the header. This applies only to a response type batch TRANS_TYPE=FRE.

Sub-key	Description
BATCH_START_END	<p>This functions similar to BATCHSEND in OUTBOUND_BATCH. If the value of BATCH_START_END is E, then it forces the creation of a batch ending with the current message.</p> <p>This is a useful feature to create response batches. The response batch must correspond to the inbound batch. That is, it contains one response detail segment for each transaction in the inbound batch. Forcing batch creation after the last message of an inbound batch satisfies this requirement.</p>

To properly generate a response batch from a request batch split, implementation engineers should follow these instructions:

When splitting a request batch, each outbound message is copied to preserve the DRIVERCTL, then translated. If the translation finds an error, then the message has an SDT segment describing the error; if there is no error, then the message is blank. That is, there is nothing in the data field but it still has the DRIVERCTL from the original message.

In this way, the last message that is blank, or contains an SDT segment, has the BATCH_START_END field in DRIVERCTL set to E. The engine knows that is the end of the messages and it is time to kick out the completed batch response message.

Message definition files

The message definition files are similar to those used for UN/EDIFACT. The NCPDP Formulary and Benefit files are stored in an ncpdpfab directory tree under formats.

These files are used to define the format:

- `datatypes`: Primitive data types for fields.
- `fields`: One line defining each field.
- `segments.idx`: One line summarizing each segment.
- `segments/*`: One file defining each segment. The file name is the segment identifier.
- `messages.idx`: One line summarizing each message.
- `messages/*`: One file defining each message. The file name is the same as the message name.

Note: The datatypes file is in the `root\formats\ncpdpfab` folder. All others are in the `site\formats\ncpdpfab` folder.

NCPDP Formulary and Benefit standard

The F&B standard is a batch oriented format. With this format, payers can push their formulary information out to a provider, clearing house, or other consumer. The purpose is to provide periodic updated information to the consumer. There are no interactive use cases for Formulary and Benefit.

The standard presents information at the health plan level. That is, there is no information regarding individual patients. A consumer uses the patient's health plan ID to access the relevant formulary.

There are two batch messages defined:

- Formulary and Benefit Load. This is from payer to consumer.
- Formulary and Benefit Response. This is from consumer back to payer.

The response message only verifies that the load file was received and notes any errors that were found in it. The load file contains transactions representing the payer's formulary.

A message consists of records (segments) which are terminated by LF. The format can use LF or CR/LF to terminate segments, but only one character can be a terminator, which in this case is LF. The protocol driver is used on the Inbound thread to convert CR/LF to LF, so the parser only gets messages with LF.

Each segment consists of data elements (fields) which are delimited by a vertical bar "|". The standard stipulates that alphanumeric fields cannot contain the vertical bar so there is no requirement for an escape character. There is no mechanism to use other delimiters. Each segment starts with a three-character segment identifier field.

The overall message consists of a batch header segment, several transactions, and a batch trailer segment.

Each transaction consists of a header segment, which identifies the transaction type, zero or more detail segments and a trailer segment.

There can be multiple types of detail segments that are valid inside a transaction.

The data structure that is associated with a transaction is called a list. That is, a Formulary Status List is defined by:

- Formulary Status Header (FHD)
- Zero or more Formulary Status Detail segments (FDT)
- Formulary Status Trailer (FTR)

A transaction can have zero detail segments when the action is D (delete list). Otherwise, some number of detail segments is required. The action is specified by a field in the transaction header.

From the view of the system, the transactions correspond to messages. The batch is split up or constructed in the protocol thread using the EDI batch facility which basically invokes a Tcl procedure.

This format is similar to HL7 but it does not work with the HL7 format. This requires messages to start with MSH and extracts the message type from data in the MSH segment.

Data elements

Add or modify data elements within specific variants by selecting an existing variant or creating a new one.

Use **File > Open > 1.0 > variant** to use an existing variant. The selected variant's definition displays in the dialog box.

Use **File > New > 1.0** to create a new variant. Specify an appropriate name and click **OK**. The standard NCPDP Formulary and Benefit base definition displays in the dialog box.

The list box in the Defined Data Elements pane contains every element defined in the selected variant, numerically sorted. Add or delete data elements in this pane.

Note: The standard NCPDP Formulary and Benefit base definition cannot be deleted from within a variant.

Defining the data element

Use the Definition of the Selected Data Element pane to modify existing or define new data elements.

- 1 Click the element name in the Defined Data Elements pane. The selected element's definition displays in the Definition of Element pane.
- 2 For **Type** click the arrow to open a list of types.
- 3 For **Length** click the arrow buttons to select the maximum number of characters that one instance of the data element can occupy in a message.
Select **Unlimited** for an unlimited field length.
- 4 For **Minimum Length** click the arrow buttons to select the minimum length of the added or edited field.

Data segments

A data segment is a logical grouping of data elements or composite data structures that are related to a specific type of data. For example, ADR Address, IDE Identity, PTY Priority. Each data segment consists of a segment identifier and one or more composite data structures or data elements.

The list box in the Defined Data Segments pane contains every segment that is defined in the selected variant, alphabetically sorted. Add or delete data segments in this pane.

Note: The standard NCPDP Formulary and Benefit base definition cannot be deleted from within a variant.

Selecting a data segment

For an existing data segment, click the segment name or specify a search string in the search text box at the bottom of the pane.

Press **Enter** or click **Search** (binoculars).

- 1 For a new data segment, click **New**.
- 2 For **Data Segment Number**, specify a unique three-letter combination for the segment number.
- 3 For **Data Segment Name**, specify a description for the segment.
- 4 Click **OK**. The new segment displays in the Defined Data Segments pane.

Defining the data segment

Use the Definition of the Selected Data Segment pane to modify existing or define new data segments. Modify a data segment definition by editing, deleting, or adding new elements to the segment.

The list box lists the data elements and composite data structures included in the data segment.

- 1 Click the segment name in the Defined Data Segments pane. The selected segment definition displays in the Definition of the Selected Data Segment pane.
- 2 Designate where to place the new data element or composite by clicking the **Add/Paste New Elements** arrow and making a selection. For example, to add an element or composite after or before a particular element or composite on the list:
 - a Click the element where the insertion is to be made.
 - b Click **After Selected Element(s)** or **Before Selected Element(s)**.
- 3 Click the **Add** tool that is located below the Definition list.
- 4 If necessary, then use the editing tools by clicking the appropriate tool that is located below the Definition of the Selected Data Segment list box.
- 5 Specify if the segment's elements are required or conditional:
 - **Mandatory** specifies the element's inclusion is required.
 - **Conditional** specifies the element's inclusion is conditional.

Messages

An NCPDP Formulary and Benefit message is defined as a single transaction set.

The list box in the Defined Messages pane contains every message defined in the selected variant, alphabetically sorted.

Note: The standard NCPDP Formulary and Benefit base definition cannot be deleted from within a variant.

Selecting a message

For an existing message, click its name or specify a search string in the search text box at the bottom of the pane.

Press **Enter** or click **Search** (binoculars).

- 1 For a new message, click **New**.
- 2 For **Message Name**, specify a unique six-letter combination for the message name.
- 3 For **Message Description**, specify a descriptive name for the message.
- 4 Click **OK**. The new message displays in the Defined Messages pane.

Defining the message

Use the Definition of the Selected Message pane to modify existing or define new messages. Modify a message layout by editing, deleting, or adding new segments to the message definition.

When a new message is added, the NCPDP Formulary and Benefit Configurator automatically inserts the required segments into the message.

- 1 Click the message name in the Defined Messages pane. The selected message's segments display in the Definition of the Selected Message pane.
- 2 Designate where to place the new segments by clicking the **Add/Paste New Segments** arrow and making a selection. For example, to add a segment or segments after or before a particular segment on the list,
 - a Click the segment where the insertion is to be made.
 - b Click **After Selected Segment(s)** or **Before Selected Segment(s)**.
- 3 Click the **Add** tool that is located below the Definition list. Use this dialog box to select the new segments.
 - a Click **List**
 - b Click and double-click the segment to add. **OK** to place the segment on the list at the location that was chosen under **Add/Paste New Segments**.
- 4 If necessary, then use the editing tools by clicking the appropriate tool that is located below the Definition of the Selected Message pane.
To modify existing segments:
 - For a single segment, click the segment to highlight it.
 - For a group of segments, click the first segment in the group and then shift-click the last segment. This selects the entire group.
- 5 Specify if the message's segments are optional or repeating:
 - **Optional** specifies the segment's inclusion is optional.
 - **Repeats** specifies whether the segment repeats.

NCPDP Formulary and Benefit format definition example

This example shows how to define an NCPDP Formulary and Benefit variant by creating a new variant containing a new field and segment.

You start by naming the variant. Select **File > New > version**. The **Create New Variant** dialog box is displayed.

Specify an appropriate name and click **OK**.

By default, after creating a variant you are working with that variant. If you exit the tool, then the next time you restart the tool you must specify the variant with which to work.

Define a NCPDP Formulary and Benefit new element

- 1 On the **Data Elements** tab, click **New** to open the **Create New Data Element** dialog box.
- 2 Specify the **Data Element Number** and **Data Element Name**.

Begin user-defined elements with 95 to 99 followed by two user-chosen numbers.

- 3 Click **OK**. This adds the new data element to the list.
- 4 In the Definition of the Selected Data Element pane, select the **Type**, **Length**, and **Minimum Length** for the new data element.

Define a new NCPDP Formulary and Benefit data segment

- 1 Click the **Data Segments** tab.
- 2 In the Defined Data Segments pane, click **New**.
- 3 Specify the **Data Segment Number** and **Data Segment Name**.
- 4 Click **OK**. This adds the new data segment to the list.
- 5 Click **Add/Paste New Elements** to select the location to place the new element. Click **Add** in the Definition of the Selected Data Segment pane to add the new data element to the segment.
- 6 Click **List** and double-click the new data element.
- 7 Click **OK**. The new segment or composite is added to the definition.
- 8 Repeat Steps 5 to 7 until all necessary elements are added.
- 9 Select the new segment in the Definition of the Selected Data Segment pane.
- 10 Select **Mandatory** or **Conditional** for each element.

Modify the base NCPDP Formulary and Benefit message layout

- 1 Click the **Messages** tab.
- 2 Click the message in which to place the new segment in the Defined Messages pane. Its segments display in the Definition of the Selected Message pane.
- 3 Click **Add/Paste New Segments** to select the location to place the new segment. Click the **Add** tool in the Definition of the Selected Message pane. The **Add Segment to Message** dialog box is displayed.
- 4 Click **List** and double-click the new segment.
- 5 Click **OK**. The new segment is added to the definition.
- 6 In the Definition of the Selected Message pane, select **Optional** or **Repeats**. This makes the new segment an optional or repeatable item in the message layout.
- 7 Select **File > Save**.
- 8 Select **File > Exit** to exit the IDE. You can also right-click the **NCPDP Formulary and Benefit** tab and select **Close**. This closes the NCPDP Formulary and Benefit Configurator and remain in the IDE.

NCPDP Telecom Configurator

NCPDP (National Council for Prescription Drug Programs) defines a set of standard message formats used in the prescription drug industry. These formats address messages from pharmacy to payer, prescriber to

pharmacy, and others. The NCPDP Telecom formats that have been designated as HIPAA standard transactions are supported.

Note: NCPDP Telecom requires a license to operate. It is activated with one of two possible license keys. An individual `cl-mm-ncpdp` license key activates NCPDP Telecom. An overall `cl-mm-ncpdpa11` license key activates all of the NCPDP formats (Telecom, Formulary and Benefit, and SCRIPT).

A basic aspect of HIPAA compliance is support for the message formats that are designated as standard. The HIPAA rules state: "No plan may refuse to accept or significantly increase the processing time for a transaction submitted in a HIPAA standard format." Therefore, these standard formats must be supported to claim HIPAA compliance.

HIPAA has designated two NCPDP Telecom formats as standard:

- Telecommunications Standard v5.1
A variable format that is designed for real-time communications between a pharmacy and a payer.
- Batch Standard v1.0 and v1.1
A file format that is used to encapsulate a set of transactions. The batch standard is a wrapper around a set of transactions that are formatted according to the Telecom v5.1 standard. HIPAA specifies that data records in the batch are in Telecommunications v5.1 format.

Copying and pasting across variants

Fields, Composite Data Structures, Tokens, and Transactions can be copied and pasted across variants. When one of these is highlighted, right-clicking opens a with **Copy**, **Paste**, and **Delete** options.

Multiple selections are permitted.

Note: The **Paste** action does not apply between different HMD formats.

NCPDP Telecom message structure

An NCPDP Telecom message consists of a group of segments that are composed of a group of fields. The first segment of every transmission, request or response, is the header segment. This does not have a segment identification, because it is a fixed field and a fixed-length segment.

Caution: Avoid changing transaction headers, as this can cause engine and Translation Tool problems.

After the header segment, other segments are included, according to the particular transaction type.

Every other segment has an identifier to denote the particular segment for parsing. Segments may display in any order after the header segment, according to whether the segment happens at the transmission or transaction level. Segments cannot repeat within a transaction. In a multi-transaction transmission, a segment may happen only once.

All fields are required positionally and filled to their maximum designation in the header segment. This is a fixed segment. If a required field is not used, then it must be filled with spaces or zeros. The fields within the header segment do not use field separators.

Other segments may have both required and optional fields. Optional fields in a segment are submitted after the required fields. Both types of fields must be preceded by a field separator and the field's identifier. Optional fields may display in any order except for those designated with a qualifier or in a repeating group. The required and optional fields may be truncated to the actual size that is used.

Parsing is accomplished with the use of separators:

- Segment Separator: Hex 1E
- Group Separator: Hex 1D
- Field Separator: Hex 1C

The general structure of a request is:

```
Header Segment (00)
Patient Segment (01)
Insurance Segment (04)
Transactions (up to four per transmission)
Claim Segment (07)
Pharmacy Provider Segment (02)
Prescriber Segment (03)
Coordination of Benefits/Other Payments Segment (05)
Workers' Compensation Segment (06)
DUR/PPS Segment (08)
Pricing Segment (11)
Coupon Segment (09)
Compound Segment (10)
Prior Authorization Segment (12)
Clinical Segment (13)
```

The number within the parentheses represents unique segment numbers for transaction.

This is an example of an NCPDP Telecom B1 message. In this example, after segment 07, both segment 02 or segment 03 can display in any order.

```
006650B112345678901074563663bbbbbb1997091598765bbbb<1E><1C>AM01<1C>CX01<1C>CY123456789<1C>C419620615<1C>C51<1C>CA
JOSEPH<1C>
CBSMITH<1C>CM123 MAIN STREET<1C>CNMY TOWN<1C>CO
CO<1C>CP34567<1C>CQ2014658923<1C>C70<1C>CX50Z123<1C>1C2<1E><1C>AM04<1C>
C2987654321<1D><1E><1C>AM07<1C>EN1<1C>D21234567<1C>E103<1C>D700006094228<1C>E730000<1C>D30<1C>D530<1C>D61<1C>
D80<1C>DE19970920<1C>DF05<1C>DJ1<1C>DK0<1C>ET30000<1C>C82<1C>DT1<1C>28EA<1E><1C>AM02<1C>E939359<1E><1C>AN03<1C>EZ8<1C>
DB00G234<1C>1E10<1C>DR
JONES<1C>PM2013639572<1C>2E1<1C>DL123456<1C>H5101<1C>4EWRIGHT<1E><1C>AM11<1C>D9557
{<1C>DC100{<1C>DX100{<1C>H71<1C>H81<1C>H9150{<1C>DQ807{<1C>DU807{<1C>DN3
```

For example, both are valid:

```
<1E><1C>AM07..<1E><1C>AM02..<1E><1C>AM03..
<1E><1C>AM07..<1E><1C>AM03..<1E><1C>AM02..
```

The same rule applies to the optional fields within segment.

Java UPoC Class

NcpdpM class is a sub-class for GRM class in Java UPOC.

```
public class NcpdpM extends Grm {
    NcpdpM (CloverEnv x);
    public static void bulkcopy (NcpdpM src, NcpdpM dest);
}
```

Message syntax

Segments and optional fields can happen in any order. To support this random order in NCPDP Telecom messages, the BNF (Backus Naur Form) message definition syntax includes a notation called Set.

A set is represented by an opening and closing pair of parentheses. A segment set indicates that any segment can display in any order and only displays at most once.

For example:

```
07
  (
    02
    03
  )
09
```

In this example, both segment sequences of 07 02 03 09 and 07 03 02 09 are valid. A segment sequence of 07 02 02 09 is invalid.

Because a segment in a set can only display one time, you can simplify the GRM addressing to not increase the group count. The GRM addresses for each segment in the above message definition, is:

```
07          -- 0(0).07
(
  02        -- 0(0).02
  03        -- 0(0).03
)
09          -- 0(0).09
```

If the set definition changes as a repeating group, then the GRM address for the above message definition is:

```
07          -- 0(0).07
{
  02        -- 1(0).02
  03        -- 1(0).03
}
09          -- 2(0).09
```

This is an example of Set notation that represents an NCPDP Telecom message type including all segments and all fields in segment 05.

```
00
[01]
[04]
{[
  07
  (
    02
    03
    [ - to start the 05 (COB) group
      AM - segment identifier
      4C - COB count
      {
        5C - other payer coverage type
        (
          6C - other payer ID qualifier
          7C - other payer ID
          E8 - other payer date
          [
            HB - other payer amount paid count
            {
```

```

    )
    ]}
    09
    10
    12
    13
    06
    08
    11
    ]
    }
    )
    [
    ]
    }
    HC - other payer amount paid qualifier
    [DV] - other payer amount paid
    5E - other payer reject count
    {6E} - other payer reject code

```

NCPDP Telecom TrxID determination

Transaction ID determination depends upon the message format. For NCPDP Telecom, the transaction ID consists of a transaction code and transmission type. For example, B1-REQUEST is generated for a billing request message and B1-RESPONSE is generated for the corresponding response.

Every NCPDP Telecom transaction has a REQUEST and RESPONSE form. Messages that are split from branches have additional fields from the batch header that are included in the transaction ID.

The included fields are Sender ID, File type (for example, text/production) and Receiver ID.

An underscore separates these fields in the transaction ID string.

The TrxID uses the form:

```
mm_tt_sender_ft_receiver
```

- `mm` is the message type (for example, B1).
- `tt` is the transmission type from the Transaction Header (T, R, or E).
- `sender` is the sender ID from the Transaction Header.
- `ft` is the file type from the Transaction Header (P or T).
- `receiver` is the receiver ID from the Transaction Header.

For example:

```
B1-REQUEST_T_SENDERID_P_RECEIVERID
```

Note: For v1.0 batches, the receiver must be omitted from the TrxID.

GRM tcl extensions

The `grmcreate` Tcl extension also accepts NCPDP Telecom.

```
grmcreate ?-msg msgId? ?-warn var? type args
```

- `type` is NCPDP.
- `args` is `vers` or `variant` or `msgType`.
 - `vers` is the version. For example, NCPDP5.1.
 - `variant` is the variant name. For example, `var1` or `{}`.
 - `msgType` is the NCPDP message type. For example, E1.

All of the GRM Tcl extensions work with NCPDP-type message handles created by `grmcreate`.

Segments

Segment identifier numbers and descriptions are:

- REQ: Request Transaction Header
- RES: Response Header
- TRN: Transaction Start
- 01: Patient
- 02: Pharmacy Provider
- 03: Prescriber
- 04: Insurance
- 05: Coordination of Benefits/Other Payments
- 06: Worker's Compensation
- 07: Claim
- 08: DUR/PPS
- 09: Coupon
- 10: Compound
- 11: Pricing
- 12: Prior Authorization
- 13: Clinical
- 20: Response Message
- 21: Response Status
- 22: Response Claim
- 23: Response Pricing
- 24: Response DUR/PPS
- 25: Response Insurance
- 26: Response Prior Authorization

Header segments do not have any identifiers defined under NCPDP Telecom. These are identified by being the first segment in a transmission.

TRN is not an NCPDP Telecom segment. This is used to start a transaction.

All other segments are designated by the standard NCPDP Telecom identifier preceded by s. For example, the segment 01 is designated s01. Segment names must contain at least one letter.

Field identifier and name

A field identifier is three digits followed by a two-character code, for example, 548-6F.

Field names must contain at least one letter. Currently, this applies only to field 28 (referred to as F28).

Note: There are several hundred fields used in NCPDP Telecom. These are defined in the data dictionary.

NCPDP Telecom tokens

The token term was created to avoid confusion with other terms that may be meaningful in an NCPDP Telecom context.

Tokens have these characteristics:

- Tokens refer to grammatical elements in an NCPDP Telecom message definition, and are generally equivalent to an NCPDP Telecom field.
- They can also refer to an NCPDP Telecom header segment, which contains multiple fields.
- The header tokens are names REQ for requests and RES for responses.
- The TRN token indicates the start of an NCPDP Telecom transaction.
- Other tokens indicate grouping, and include the curly brace, parenthesis, and square bracket.

NCPDP Telecom batch standard

The NCPDP Telecom Batch Standard uses a file layout format that is used to encapsulate NCPDP Telecom transactions.

The batch file consists of Header, Data, and Trailer sections.

Transaction header and trailer

The header and trailer must be present in every transmission:

```
Transaction Header 1 per file
    Transaction Detail Data
    Transaction Detail Data
    . . . up to 9,999,999,997 records per file
Transaction Trailer 1 per file
```

The Transaction Header and Transaction Trailer segments are fixed-length segments. The Transaction Detail Data segment contains all of the NCPDP Telecom transaction.

Batch standard v1.0 and v1.1 differences

This table shows the differences between version 1.0 and 1.1 batch headers:

Version 1.0 batch header	Version 1.1 batch header
The Batch Number field is 5 characters long.	The Batch Number field is 7 characters long.
There is no receiver ID field immediately preceding the last End-of-Text character.	There is a receiver ID field of 24 characters immediately preceding the last End-of-Text character.

This table shows the difference between version 1.0 and 1.1 batch trailers:

Version 1.0 batch trailer	Version 1.1 batch trailer
The Batch Number field is 5 characters long.	The Batch Number field is 7 characters long.

NCPDP Telecom batch transaction example

```
(STX)00T1234567890123456789022229876747199412011632T11ABCDEFGHYTEWQASDXZAQ1234(ETX)
(STX)G110000543219999993201.....(ETX)
(STX)G110000543229999993201.....(ETX)
(STX)99987674700000000004Initial Test(ETX)
```

Batch processing

Processing is handled by standard Tcl procedures in the thread. This table shows the driver control keys that are used to control these procedures.

{BATCHTYPE NCPDP}	Identifies the current message as an outbound accumulating batch.
{TRANSACTIONID nnn}	Sets the transaction control number programmatically. If no control number is specified, then the default is to start at one (1) and increment.
{BATCHID nnn}	Sets the batch control number programmatically. If no control number is specified, then the default is to use a counter.
{BATCHSIZE nnn}	Programmatically controls the maximum batch size. The default would be to use a value stored in NetConfig.
{BATCH SEND}	When a message happens with this key set, it indicates that a batch should be generated. This would typically be under the control of a timer.

The outbound procedure examines the driver control for the {BATCHTYPE NCPDP} key.

If it finds this key, then it holds on to the message by not supplying a disposition. This leaves the message in the recovery database. The message handle is stored in a global variable for later use.

If an outbound message does not have this key, then it is a transaction that should be accumulated in the outbound batch.

If no outbound batch exists, then one is created.

After a message is accumulated in the outbound batch, the new batch is released. The new batch has a disposition of `OVER` which places it in the inbound queue. The original message is stopped. The inbound procedure takes a message that contains `{BATCHTYPE NCPDP}` and gives it a disposition of `OVER`. This returns it to the outbound queue where it waits for more messages to be accumulated. This ensures that the batch is always properly recovered. When there is a crash or thread that is stopped then started, the batch messages are redelivered to outbound processing and re-held.

A problem happens, though, when two messages are processed outbound before the batch message is re-delivered to the outbound procedure. The best method of handling this is to permit the processing of another `{BATCHTYPE NCPDP}` message, which flows back around. The outbound procedure continues to remember the original batch handle. Any batch message that arrives and is not the currently remembered batch must be appended to the original batch. The original batch gets `OVER` and the later batch is stopped. This technique ensures that messages remain in order both for normal processing or recovery.

The stored batch is sent out with a `CONTINUE` disposition. This is sent when the maximum number of messages is received or a `{BATCH SEND}` message is received.

Messages accumulating in a batch must be stored in the recovery database if recovery is enabled.

Inbound NCPDP Telecom batches

Inbound NCPDP Telecom batches are supported with an EDI Batch attribute for inbound threads. If EDI Batch type NCPDP Telecom is specified for an inbound thread, then all inbound messages are treated as batches.

Each inbound message is split into individual transactions for processing. Values from the batch header and trailer are attached to the individual transaction metadata.

For example:

```
(STX)00T1234567890123456789022229876747199412011632T11ABCDEFGHYTEWQASDXZAQ1234(ETX)
(STX)G110000543219999993201.....(ETX)
(STX)G110000543229999993201.....(ETX)
(STX)9998767470000000004Initial Test      (ETX)
```

Fields

A field represents a specific piece of data, such as patient name, address, or doctor ID, and has a specific set of attributes.

The list box in the Defined Fields pane contains every field defined in the selected variant, numerically sorted. Add or delete fields in this pane.

Add or modify fields within specific variants. You can select an existing variant or create a new one.

Note: The standard NCPDP Telecom base definition cannot be deleted from within a variant.

Selecting a field

Use **File > Open > version > variant** to use an existing variant. The selected variant's definition displays in the dialog box.

Use **File > New > version** to create a new variant. Specify an appropriate name and click **OK**. The standard NCPDP Telecom base definition displays in the dialog box.

- 1 For an existing field, click the field name or specify a search string in the search text box at the bottom of the pane. Press **Enter** or click **Search** (binoculars).
- 2 For a new field, click the **New** button.
 - For **Field Number**, specify two characters that are unique among all fields in the current variant.
 - For **Field Name**, specify a descriptive name for the field. Unique field names are not necessary.
 - Click **OK**. The new field displays in the Defined Fields panel.

Defining the field

Use the Definition of the Selected Field pane to modify existing or define new fields.

- 1 Click the field name in the Defined Fields pane. The selected field's definition displays in the Definition of the Selected Field pane.
- 2 Edit the field's attributes as required.
 - For **Type**, click the arrow to open a list of types.
 - For **Length**, click the arrow buttons to select the maximum number of characters that one instance of the field can occupy in a transaction. Select **Unlimited** for an unlimited field length.
 - For **Minimum Length**, click the arrow buttons to select the minimum length of the added or edited field.

Composite data structures

A field represents a specific piece of data, and a composite data structure consists of two or more fields.

Tokens

The list box in the Defined Tokens pane contains every token defined in the selected variant, alphabetically sorted. Add or delete tokens in this pane.

Note: The standard NCPDP Telecom base definition cannot be deleted from within a variant.

Selecting a token

- 1 For an existing token selection, click the token name or specify a search string in the search text box at the bottom of the pane. Press **Enter** or click **Search** (binoculars).
- 2 For a new token, click **New**
 - For **Token Number**, specify a unique combination for the token number.
 - For **Token Name**, specify a description for the token.
 - Click **OK**. The new token displays in the **Defined Tokens** list box.

Defining the token

Use the Definition of the Selected Token pane to modify existing or define new tokens. Modify a token definition by editing, deleting, or adding new elements to the token.

The list box lists the elements included in the token.

- 1 Click the token name in the Defined Tokens pane. The selected token definition displays in the Definition of the Selected Token pane.
- 2 Designate where to place the new element by clicking the **Add/Paste New Elements** arrow and making a selection. For example, to add an element after or before a particular element on the list:
 - a Click the element where the insertion is to be made.
 - b Click **After Selected Element(s)** or **Before Selected Element(s)**.
- 3 Click the **Add** tool that is located below the Definition list. Use this dialog box to select the new element or composite.
 - Click **List** and double-click the element to add.
 - Click **OK** to place the element or composite on the list at the location that was chosen under **Add/Paste New Elements**.
- 4 If necessary, then use the editing tools by clicking the appropriate tool that is located below the **Definition of the Selected Token** list box.
- 5 Specify if the token's elements are **Mandatory**, **Optional**, or **Conditional**.

Transactions

The list box in the Defined Transactions pane contains every transaction defined in the selected variant, alphabetically sorted.

Note: The standard NCPDP Telecom base definition cannot be deleted from within a variant.

Selecting a transaction

- 1 For an existing transaction, click its name or specify a search string in the search text box at the bottom of the pane. Press **Enter** or click **Search** (binoculars).

- 2 For a new transaction, click **New**.
 - For **Transaction Name**, specify a unique two-character transaction code followed by a usage designator
 - For **Transaction Description**, specify a descriptive name for the transaction.
 - Click **OK**. The new transaction displays in the Defined Transactions pane.

Defining the transaction

Use the Definition of the Selected Transaction pane to modify existing or define new transactions. Modify a transaction layout by editing, deleting, or adding new tokens to the transaction definition.

- 1 Click the transaction name in the Defined Transactions pane. The selected transaction's tokens display in the Definition of the Selected Transaction pane.
- 2 Designate where to place the new token by clicking the **Add/Paste New Tokens** arrow and making a selection. For example, to add a token after or before a particular token on the list:
 - a Click the token where the insertion is to be made.
 - b Click **After Selected Token(s)** or **Before Selected Token(s)**.
- 3 Then, click the **Add** tool that is located below the Definition list.
Use this dialog box to select the new tokens.
 - Click **List** and double-click the token to add.
 - Click **OK**. This places the token on the list at the location that was chosen under **Add/Paste New Tokens**.
- 4 If necessary, then use the editing tools by clicking the appropriate tool that is located below the **Definition of the Selected Transaction** list box. To modify existing tokens:
 - For a single token, click the token to highlight it.
 - For a group of tokens, click the first token in the group and then shift-click the last token. This selects the entire group.
- 5 Specify if the transaction's tokens are optional or repeating:
 - **Optional** specifies the token's inclusion is optional.
 - **Repeats** specifies whether the token repeats.
If the token is repeating, then select the **No Limit** check box or click the arrows to select the number of times the token repeats.
 - **Set** specifies that the token can display in any order only once.
 - **No Limits** specifies an unlimited number of repeats.

NCPDP Telecom format definition example

You start by naming the variant. Select **File > New > version**. The **Create New Variant** dialog box is displayed. Version NCPDP5.1 of NCPDP Telecom is supported.

Specify an appropriate name and click **OK**.

The standard NCPDP Telecom base definition displays in the dialog box.

Define a NCPDP Telecom field

- 1 Click the **Fields** tab.
- 2 In the Defined Fields pane, click **New**.
- 3 Specify the **Field Number** and **Field Name**. Specify two characters that are unique among all fields in the current variant. Then, specify a descriptive field name. Unique field names are not necessary.
- 4 Click **OK**.
- 5 In the Definition of the Selected Field pane, select the **Type**, **Length**, and **Minimum Length** for the new field.

Define a NCPDP Telecom token

- 1 Click the Tokens tab.
- 2 In the Defined Tokens pane, click **New**.
- 3 Specify the **Token Number** and **Token Name**. Specify a unique combination for the token number, and a descriptive token name.
- 4 Click **OK**.
- 5 Click **Add/Paste New Elements** to select the location to place the new field. Click the **Add** tool in the Definition of the Selected Token pane to add the new field to the token.
- 6 Click **List** and double-click the new field.
- 7 Click **OK**. The new token or composite is added to the definition.
- 8 Repeat Steps 5 to 7 until all necessary fields are added.
- 9 Select the new token in the Definition of the Selected Token pane.
- 10 Select **Mandatory**, **Optional**, or **Conditional** as required for each token.

Modify the NCPDP Telecom base transaction layout

- 1 Click the Transactions tab.
- 2 Click the transaction in which to place the new token in the Defined Transactions pane. Its tokens display in the Definition of the Selected Transaction pane.
- 3 Click **Add/Paste New Tokens** to select the location to place the new token.
Note: Do not paste a new token directly before the RES, REQ, or TRN tokens.
- 4 Click the **Add** tool in the Definition of the Selected Transaction pane.
- 5 Click **List** and double-click the new token.
- 6 Click **OK**. The new token is added to the definition.
- 7 In the Definition of the Selected Transaction pane, select **Optional** or **Repeats**. These make the new token an optional or repeatable item in the transaction layout.

- To repeat, select the **No Limit** check box or specify the number of times in the field below the **No Limit** check box.
 - Select the **Set** check box to create a set in the transaction layout. Set is similar to a repeat with optional fields. The main difference is that no additional repeats are generated, so iteration is not required. A Set is represented by an opening and closing pair of parentheses. A segment set indicates that any segment can display in any order and only displays at most once.
- 8 Select **File > Save**.
 - 9 Select **File > Exit** to exit the IDE. You can also right-click the NCPDP Telecom tab and select **Close** to close the NCPDP Telecom Configurator and remain in the IDE.

Network Configurator

Use the Network Configurator to define and modify network connections, and the actions to take when handling communication transactions flowing across those connections.

Configure all connections from the viewpoint of the system site, which is assumed to be the starting point for any connection.

The complete system configuration includes:

- The connection's physical description, including low-level protocol, data format, and message management specifications.
- Attribute settings to define the thread managing the connection.

Working with threads

Using the IDE through a remote desktop could take a longer time to repaint when there are complex threads. This is especially so on a system with anything but a high bandwidth connection,

To help speed the process of configuration, you can specify a uniform colored background. This selection is made by selecting **Use uniform colored background** on the **NetConfig/NetMonitor** tab of the **Options > Client Preferences** dialog box. By default, this is set to **Use Uniform colored background**.

Process and thread name limits

There is an engine limit of 57 characters for process names and 64 characters for thread names.

These limits are checked by the engine and `hcinetcheck`. `hcinetcheck` checks the Network Configurator file for configuration errors.

If the limits are exceeded, then the engine does not run or validate.

Note: To stay under these limits, keep the process and thread names under 48 characters.

Basic network configuration

Network Configurator is used to create and modify network connections and the threads that manage them.

Each icon in the dialog box consists of a bitmap and name that represents a single connection controlled by an engine thread.

The engine can contain several processes that contain several threads. To the user, these processes are handled transparently.

Configure each connection completely, or add a series of connections at one time as a rough layout of the system. You can enter the details of the configuration information later.

Note: A maximum of one (1) Network Configurator can run on the IDE at any given time.

Backward compatibility

The current IDE can open older NetConfigs, but always saves in the current root version, adding all new keys and configurations.

User interface

The Network Configurator uses a "route-oriented approach" to configure routing behaviors for each thread. The GUI is a graphical representation that shows message or reply routes.

To create a thread with a particular protocol, drag the protocol icon onto the Network Graph panel. You can also right-click the panel to create a new thread.

After highlighting the thread, on the configuration toolbar click **Configure Thread Routes**. This switches to the route configuration screen for the currently selected thread.

Property panel

This is a container of property detail tabs for the currently selected threads or routes on the Network Graph panel.

Clicking **Verify Thread** validates the currently selected configuration.

- Closing a tool or saving a file triggers validation; if there are errors, then it prompts for confirmation.
- Clicking the Verification tab lists any errors.
- Non-critical errors prompt a warning and can be saved.

For flexibility, the panel can be docked on any of the GUI's four sides. To change the Property panel's location, click and hold a tab and drag it to any side.

A shaded box indicates the location. Release the mouse and the panel is shown on the selected side.

Critical errors must be fixed before you can save the changes. For example, selecting the **Save outbound messages** check box without entering a file name and then clicking **Apply** prompts an error message. Examples of critical errors include:

- Selecting the Fileset FTP protocol for a thread without configuring the Host and User fields.
- In a route configuration view, any errors that are related to configuration integrity are critical errors. These must be fixed before applying the changes or moving to other nodes. For example, adding an Xlate node and not configuring the **Xlate** field.

Route line styles

The **Options > Client Preferences > NetConfig/NetMonitor** tab has an option to select the route line style. To ease the visual flow, you can select plain or z-style lines.

"Save As" option

When a NetConfig file is saved, it is saved under the `netconfig` directory of the site directory.

Each site's NetConfig file contains its own sub-directory, `.pni` file, and java driver.

Reading/editing master site objects from another site

With this enhancement, you can:

- Edit the master site from a site.
- Edit master site objects from other sites.
- Open master site objects as read-only from other sites.

This is useful when you have a large number of production sites.

You can also:

- Open a master site's (`$HCIMASTERSITEDIR`) HL7 variants from a site (`$HCISITE`).
- Open a variant that is located in the master site from the current site

These options and a menu that contains all entries from the master site are on the Network Configurator toolbar.

Enabling/Disabling procs in the TPS editor

When using the TPS Editor to add a TPS proc to the list, a check box is provided next to each listing. This is to enable/disable the proc. If a proc is disabled, then the proc remains on the list, but is unavailable. It is also not run when the NetConfig is saved. The args are saved with the disabled proc.

This is convenient when you have a complex thread configuration with a large number of TPS scripts and must temporarily remove and re-add those scripts. For example, when working on conversions, upgrades, or troubleshooting issues.

Thread icons

The icons on the Network Configurator represent threads, or connections, that come from or go to an external system. Each system that sends and receives messages through the engine should have a corresponding icon.

Ordered NetConfig

NetConfig contains many processes and threads. There are two types of thread:

- Protocol thread
- Destination thread

The order of processes and threads is often changed after saving. For example, in previous Cloverleaf versions, saving the NetConfig could cause the process and thread order within NetConfig to change.

Maintaining the order of processes and threads is necessary to for merging changes by multiple users.

The user ID is a unique property of threads and processes. This ID is composed of hash code containing the file name and sequence number. The sequence starts at "1" and increases by 1. For example, when there are 4 threads, the sequences are from 1 to 4.

The ID is set for each process and thread when loading NetConfig. This property cannot be edited or deleted.

Renamed threads keep their IDs, so their positions do not change. Threads and processes are sorted by ID when saving NetConfig.

New threads and processes do not have an ID, and are sorted by name. These are saved at the end of threads and processes.

Threads and processes from several NetConfig are ordered by file name first, followed by sequence.

Overwritten or merged threads and processes keep their positions.

For BOX deployment, the threads in NetConfig can be from a BOX or site. They are saved together when they are from the same file. For example:

Thread from site	Thread from BOX
thread1	thread2
thread3	thread4

After BOX deployment, the order could be:

- thread1, thread3, thread2, thread4
- thread2, thread4, thread1, thread3

Creating a new thread/destination

The right-click menu has these options:

- **New Thread**
Click to create a new thread icon. The first icon displays with the name "conn_1." Subsequent icons are numbered consecutively. Create as many icons as required for your configuration.
- **New Thread from Template**
This option is for reusing thread definitions.
See [Thread template](#).
- **New Destination**
Select this to create a destination that represents a public thread from out-of-site on the current site.

See [Inter-site routing](#).

- **Paste**

Use this option to create copies of threads:

- Right-click the icon to be copied. A connection menu opens for that thread. Select **Copy**.
- Right-click in an empty portion of the dialog box. Select **Paste Thread** from the menu.
- A duplicate icon displays numbered `conn_x1`, where `x` is the thread number and `1` is the copy number. Subsequent copies are numbered consecutively.

Configuring the icon

Right-click the icon to be configured to open a menu of options.

- **Verify**

When this is clicked, if a NetConfig error exists, the dialog box automatically goes to the Verifications tab with a listing of all errors.

- **Copy**

Generates a duplicate of the selected thread. The copy contains the same configuration characteristics as the original, including routes. The routes, including reply routes, cannot be copied or pasted alone. Copies are based on the related source/destination thread selection status:

Routes cannot be copied, even if selected, if the source or destination is not selected.

If the route and only its source thread are selected, then the route is also copied. The new pasted route connects the new copied thread to the original destination.

If the route and only its destination thread are selected, then the route is also copied. The new pasted route connects the original source to new pasted destination thread.

If only the destination thread is selected without selecting the route, then the route to the selected destination is not copied.

If the source and destination threads of a route are selected and copied, then the new source thread has a route to the new destination thread. This is without regard to the route selection status. The pasted route does not point to the original destination thread.

- **Delete**

Deletes all selected threads. The route/reply route is deleted if any of its source or destination threads are deleted.

- **Create Route**

Clicking this turns the cursor into a crosshair, so you can create a new route to another thread.

- **Create Reply Route**

Clicking this turns the cursor into a crosshair, so you can create a new reply route to another thread.

- **Create View** Opens the Create View dialog box, where you can create a view for the new thread.

- **Create BOX**

Opens the New BOX Wizard.

- **Save as Template**

Saves the selected thread as a template.

See [Thread template](#).

TPS Editor language support

Support is available for JavaScript UPoC and Python UPoC definitions. On the GUI, JavaScript and Python drop-down lists are in Network Configurator, Translation Configurator, TPS, and the TPS testing tool.

JavaScript and Python are stored in *site/scripts*. The JavaScript and Python UPoC are consistent with the Tcl UPoC support for master site and current site loading files.

Process configuration dialog box

Select **Process > Configure** to open the **Process Configuration** dialog box to the default **Properties** tab.

This table shows the available configuration items and how to configure:

Option	Description
Default Engine Log Configuration	Specify the default EO aliases to apply to this process. These default EO aliases apply to all threads in the process. Click List to select from a predefined list.
Xlate Engine Log Configuration	<p>Specify the translation thread EO aliases to apply. There is only one translation thread per process. Click List to select from a predefined list.</p> <p>The process default and xlate EO aliases are evaluated and applied in the order they are listed, from left to right.</p>

Option	Description
Process Bitmap	<p>Specify the bitmap file to use for all threads in the selected process. Or, click List to select from a file browser dialog box.</p> <p>Select Overwrite Thread Bitmap to have the Process Bitmap setting to overwrite the thread Bitmap setting. If this check box is not available, then no process bitmap has been specified.</p> <p>These settings are saved as process properties, and are reloaded to refresh the IDE when the corresponding process is selected.</p> <p>The process bitmap is used if Process Bitmap is specified and the Overwrite Thread Bitmap option is selected, or if there is no thread Bitmap specified.</p> <p>The thread bitmap is used if Process Bitmap is not specified and a thread Bitmap is specified.</p> <p>If none of the above conditions are met, then the default is the thread bitmap.</p> <p>By default, Process Bitmap is empty, the Overwrite Thread Bitmap is cleared, and the Thread Bitmap setting on the Thread Properties dialog box is empty.</p>
Error Database Procs	<p>The error database TPS gives you a control point before a message is written to the error database. This indicates you have the opportunity to change the error message content and to route the message to another context.</p>
Retries	<p>The maximum number of times to attempt message transmission:</p> <ul style="list-style-type: none"> • "0" is the initial transmission only. • "1" is the initial transmission plus one retry. • "-1" is unlimited retries.
Automatic Log Cycling	<p>This automatically cycles message logs when they get to the size that is specified in Threshold in Kilobytes. Cycling a save file causes the current file to close, renames it to <code>old_name.old</code>, and opens a new save file. The save files are located in the subdirectory named <code>%HCISITEDIR%\exec\processes\process</code>.</p>
Threshold in Kilobytes	<p>The maximum number of kilobytes the log file can attain before cycling.</p>

Option	Description
Automatic SMAT Cycling	This cycles the SMAT files. By default, this option is cleared. See "Automatic SMAT cycling conditions" in SMAT history .
Threshold in Kilobytes	The maximum number of kilobytes that the messages size in the SMAT files can attain before cycling.
Disk-Based Queuing	<p>This uses disk-based queuing for this process.</p> <p>Under normal processing, the queues in the threads and process are kept in memory (RAM) and backed by the recovery database, if in use. Disk-based queuing moves the data of the messages, but not the metadata, to disk only. This reduces the RAM required by the process.</p> <p>Use this option if there is a high volume process, or if the process is working with large messages.</p> <p>Messages using disk-based messaging save to the recovery database.</p>
Threshold in Megabytes	<p>The maximum number of megabytes the log file can attain before cycling.</p> <p>For example, disk-based queuing is selected and the total RAM that is required for the queues exceeds this threshold. In this case, all subsequent data posted to the queues is saved to the recovery database.</p> <p>After the RAM requirements fall below this value, the message data is stored in RAM.</p>

Option	Description
Translation Throttling	<p>Select this and then configure the Minimum, Maximum, or Percent values. These control how many messages the translation thread processes each time it runs. Under normal circumstances, the translation thread processes all of the messages on its queue. By specifying translation throttling, the number of messages that are processed is limited. This permits other threads in the engine process to run.</p> <p>Minimum specifies that, if the number of messages on the queue is less than this value, all of the messages are processed. If the queue has more than this many messages, then the percentage is applied.</p> <p>Maximum specifies the absolute maximum number of messages to process. If the Minimum is exceeded, then the percentage is applied. If the resulting number of messages is greater than this maximum, then only the maximum number of messages is processed.</p> <p>Percent specifies the percentage of messages to process if the queue has more than the minimum number of messages.</p> <p>Clear this for the translation thread to process all of the messages in the queue.</p>
Route replies to original source only	<p>When this option is selected, the reply messages are routed only to the original source thread.</p> <p>When cleared, the outbound thread reply is routed back by all reply routes to all inbound threads.</p> <p>Select this when you have multiple inbound threads and only require the reply to go to the originating thread.</p> <p>By default, this option is cleared.</p>
Run translation procs in start-mode	<p>This runs the translation thread Tcl procedures in startup mode when a protocol thread is started. Unlike other TPS procs (for example, inbound TPS, outbound TPS), pre- and post-xlate TPS procs are not run in startup mode. Use in situations where the startup mode is required to run on pre- and post-xlate procs. For example, initializing global variables before running procs in run mode.</p>

Option	Description
Multi-threaded Translation	The translation thread can perform all transformations within a process with a single translation thread per process, or with a multi-threaded translation per process. See Multi-threaded translation .
Reuse parsed message	<p>A parse tree created by the <code>grmcreate</code> command in an inbound TPS or another UPoC before translation is cached for use by the translation engine.</p> <p>By selecting this option, the <code>grm</code> object is cached to the message structure. In this way, the <code>grmcreate</code> Tcl command, TrxID parsing, and <code>xlate</code> parsing cache the <code>grm</code> every time it is parsed. It also cancels any subsequent parse if the message content is unmodified and the message format is the same.</p> <p>The name in the Tcl command must match the name of the <code>xlt</code> file. If the names do not match, then the parse is not reused.</p> <p>Note: This option is only supported for the XML message format; parsed data is not saved between contexts.</p>
Enable message tracing (process level)	<p>When this is selected, tracing information is saved with the message trace framework to the tracing database.</p> <p>Message tracing is enabled on the process level or thread level.</p> <p>If process level tracing is enabled, then there is no requirement to turn on tracing for each individual thread under the process. Conversely, if process level tracing is disabled, then tracing for a thread is enabled with the thread level option.</p> <p>By default, this option is cleared. When this is selected, all threads under the process automatically have tracing enabled.</p>
Minimum Free Virtual Memory	The minimum free virtual memory threshold for the process.

Process Configuration dialog box Notes tab

For information, see [Notes tab](#).

Java Driver tab: JVM options

This table shows the available JVM options:

Option	Description
Working Directory	Select the working directory for the JVM. This corresponds to the START_DIR property in the JVM section.
Browse	<p>Opens a File Chooser dialog box, where you can select a directory under the root directory.</p> <p>This dialog box converts the selected directory from absolute to relative. It does this by replacing the prefixed site directory with \$SITEPATH or \$ROOTPATH to make the protocol portable.</p>
Classpath	<p>A classpath is a list of class folders and jar files. \$ROOTPATH/lib/java/JavaDriver.jar is automatically added to the classpath and should never be deleted.</p> <p>Click Add JAR to add a jar file into the classpath. The JAR File chooser is opened when Add JAR is clicked. Then, you can specify a jar file. The top directory is the root directory.</p> <p>The Class Folder chooser is similar to the Add JAR chooser, except that you can only select a folder.</p> <p>These selections convert the selected directory from absolute to relative by replacing the prefixed directory with \$ROOTPATH, \$SITEPATH, or the working directory.</p>
Move Up and Move Down	This adjusts the sequences of the classpath entries. These are enabled when at least a classpath entry is selected in the classpath list.
Maximum/Minimum Heap Size	<p>These options configure the maximum and minimum Java heap size of the JVM. They correspond to the MEM_MIN and MEM_MAX properties in JVM section.</p> <p>The default minimum heap size is 64 MB and the default maximum is 128 MB. The permitted size is 32 ~ 1000 megabytes.</p>

Option	Description
Additional JVM Options	<p>Configurations correspond to the USER property in the JVM section. Each item in the list has a USER entry in the .pni file.</p> <p>The maximum and minimum heap size option values must be between 32 MB and 1000 MB. If these values are outside this range, then an error dialog box opens.</p> <p>The minimum heap size cannot be greater than the maximum heap size. If the minimum heap size is greater than the maximum heap size, then an error dialog box opens.</p> <p>When the minimum heap size is blank, 64 MB is the system default. If the maximum heap size is less than the default minimum heap size, then an error dialog box opens.</p> <p>When the maximum heap size is blank, 128 MB is used as the default. If the minimum heap size is greater than the default maximum heap size, then an error dialog box opens.</p>

Java Driver tab: User-defined options

User-defined options are derived from the `DEFINE` property in the JVM section. They are put into a separate tab to make the user interface more convenient when configuring the options.

Click **Add** to add an empty entry into the table.

Table entries are added to the `DEFINE` property in the JVM section. Each entry in the table has a `DEFINE` entry in the .pni file.

Thread template

To facilitate the configuration of complex thread definitions, you can use a thread template.

Use a template to set up the thread configuration. To create a new thread, instead of starting from scratch, you can specify a template that matches your requirement. Fields such as **Port** are clearly marked, for example, `<enter-port-number-here>`, for user-supplied variables.

You can also create a template from an existing thread.

The supplied templates are:

- `HL7_MLLP_TCP_INBOUND`
 - Listens for connections on a specified port

- Expects HL7 messages that are encapsulated in MLLP wrapper
- Messages starting with MSH are ACK'd with HL7 AA
- Messages not starting with MSH are NAK'd with HL7 AE
- HL7_MLLP_TCP_OUTBOUND
 - Connects to remote system on specified port and IP
 - Sends HL7 messages that are encapsulated in MLLP wrapper
 - Expects MLLP-encapsulated acknowledgment; any replies are stopped
- HL7_MLLP_TCP_OUTBOUND_VALIDATION
 - Connects to remote system on specified port and IP
 - Sends HL7 messages that are encapsulated in MLLP wrapper
 - Expects MLLP-encapsulated HL7 acknowledgment

Action is taken based on reply type:

 - AA or CA: Stop reply and send next message
 - AE or CE: Put original message in error database, stop reply, send next message
 - AR" or CR: Resend original message up to three times. If more than three AR or CR are received, then treat as AE
- CL_to_CL_INBOUND and CL_to_CL_OUTBOUND

Sets up a thread-to-thread connection.

 - Length encoded TCP/IP
 - IB has standard ACK
 - OB has recovery and ACK validation
 - Used for Query/Response connections to be created
 - Uses queuing and memory for fast hand-off
 - Permits multi-server connections
 - Faster than TCP/IP hand-off, where milliseconds are important in a Query/Response

The template settings are saved as template file under the folder `$HCIR00T/templates/NetConfig`.

The specific name has a `_template` suffix as the actual file name. For example, template file `HL7_NCK_template`.

The structure of the thread template matches that in `NetConfig`.

Selecting a template

- 1 To create a new thread from an existing thread template, right-click in an empty portion of the panel to open a menu and select **New Thread from Template**.
- 2 A menu opens with template names. Select a template to create a thread using this template.
 - When there are more than 10 templates under current root, a **More** item is added at the bottom of the menu.
 - All `NetConfig` thread templates under the current root are shown on the list. Only one template can be selected at a time.

Creating a template

- 1 Right-click a thread icon to open the connection menu.
- 2 Select **Save As Template**. This opens a **Save As Template** dialog box where you can specify a new template name.
- 3 Click **OK** to save the new template. Templates are saved under `$HCIR00T/templates/NetConfig` with a `_template` suffix.

Multi-threaded translation

An engine process has a command thread (the “scheduler”), a translation thread, and protocol threads.

The translation thread can have multiple operating system sub-threads that run outside the engine’s scheduler. It can perform all transformations within a process with a single translation thread per process, or with a multi-threaded translation per process. This permits protocol threads to run when translations are being processed. If the computer has more than one processor, then threads use the full power of the machine. The sub-threads, when working on a translation, are under the control of the operating system. These can be assigned to any available CPU or suspended as required when more important tasks must be handled.

Note: Previous NetConfigs can be read by version 6.2 and later. Version 5.8 and later of NetConfig that contains threaded or chained xlate configuration information is not compatible with previous versions.

Multi-threaded translation configuration

In the **Process Configuration** dialog box, select **Multi-threaded Translation** and click **Configure** to set up multi-threaded translations. This opens the **Multi-threaded Translation** dialog box.

When this option is not selected, the behavior is only one translation thread per process. In this case, the number of translation sub-threads is "0."

This table shows the available options in configuring a multi-threaded translation:

Option	Description
Number of Translation Sub-threads	This specifies the number of translation sub-threads to be used. Select from 1 to 16; the default value is 1.

Option	Description
Message Ordering	<p data-bbox="818 302 1409 430">This specifies how messages from a source protocol thread are ordered. This option is unavailable when the number of translation sub-threads is equal to 1; otherwise, it is enabled.</p> <ul data-bbox="818 443 1409 808" style="list-style-type: none"> • Any (default): It is unnecessary for messages taken from the pre-xlate queue to match the order they exit the xlate thread. In general, this gives the best performance in such mode. • Ordered. Messages from a source protocol thread must be maintained exactly, so only one message from a source is processed at any given time. Usually, this mode is specified only when the xlate Tcl procedures are maintaining some global variables which require that those messages should be in order. <p data-bbox="867 823 1409 1052">This is source-ordered and applies to the sequence of messages from a single source protocol thread. If there are several sources and more than one sub-thread, then overlap processing can happen. This is because translations for more than one source can be active simultaneously.</p> <p data-bbox="867 1066 1409 1194">Using this value can effect performance when most of the messages are from a single source. This is because the engine operates as if there were no sub-threads.</p> <ul data-bbox="818 1209 1409 1438" style="list-style-type: none"> • Thread Out: Messages from a source protocol thread can be started at any time but the output of the translation must match the input order. Note that this is source ordering as it only maintains the order of messages from a single source. No order is maintained for messages from several sources. <p data-bbox="867 1453 1409 1619">This permits several sub-threads to be processing messages from the same source simultaneously. The routing from the translation is in exactly the order they are taken from the preXlate queue.</p> <p data-bbox="867 1633 1409 1761">Use this setting if you have no Tcl procedures in the xlate that require ordering but the order of messages from the xlate thread must be maintained.</p> <p data-bbox="818 1785 1409 1845">Whichever type is selected, its respective help information is shown in the text area.</p>

Option	Description
All Restricted (default)	<p>You can restrict the use of Tcl interpreters by clicking one of the Tcl Interpreters option buttons.</p> <p>These options are used to specify how Tcl interpreters are selected. This is because if in your Tcl procedures globals are created and used, the same interpreter must be used for every translation.</p> <p>In addition to the Tcl procedures that are associated with an xlate, there are Tcl fragments that can be used in the actual translations. These are used to set or modify the values placed in the output of the translation. When fragments are run by Tcl, the same interpreter is used as other Tcl user calls in the xlate processing.</p> <p>All xlates use the same Tcl interpreter independent of the source or type of the message.</p> <p>Thus at any given time, there can be only one Tcl procedure running. Tcl procedures are called in the start and shutdown modes.</p> <p>This is the safest mode if you are not absolutely sure about the Tcl that is used in translation. This gives the worst performance.</p> <p>Because <code>xlateInVals</code> and <code>xlateOutVals</code> are global variables, All Unrestricted is usually the best performance option to use when doing multi-threaded translates.</p>
All Unrestricted	<p>No Tcl run that is in the xlates uses any global. The interpreter that is used for the translation of a message can be any that are available independent of the message source.</p> <p>The start and shutdown mode calls to the Tcl procedures are not called in this mode.</p> <p>In this case, an interpreter that is specific to the sub-thread doing the translation is used for the procedure calls. The interpreter that is used can vary from message to message.</p> <p>Java UPoCs can only use the unrestricted Tcl mode if the number of sub-threads in xlate is not zero.</p>

Option	Description
Per Source	<p>Each source of a data message gets an interpreter for xlate processing.</p> <p>This is used for all Tcl user-defined calls related to the message. In the case of a reply message, the source is considered to be the original source thread to which the reply is connected. The start and shut-down mode calls are made for each procedure before it is used in the translation.</p> <p>This interpreter is used in case you set globals for the outbound message that is required to process the reply. Note that this setting is applied only if the number of sub-threads is not zero. This ensures that zero sub-thread processing is exactly the same as the previous version of the xlate thread.</p>
Unrestricted Source List	<p>A Tcl list of message sources that are to be unrestricted as to which interpreter is used. Message sources not in the list are restricted to a single interpreter independent of the message source. The text field and List are available only when this option is selected.</p> <p>Using this value can yield the best performance if the use of Tcl globals varies by message source.</p> <p>When this option is selected, clicking List opens a Threads dialog box. This lists all threads that are under the selected process. On this dialog box, you select the threads as message sources that are not to be restricted to which interpreter is used if required. Selected threads display in the text field.</p>

Message tracing

There are many situations where it may be useful for you to trace a message's path through the engine. For example, when troubleshooting a message that is not going where expected, or debugging a route that is not behaving as expected.

Message tracing can:

- Let you see what messages enter the engine and the resultant messages that go out.
- Take an outbound message and see from what inbound message it derived.
- Take a message and see what reply messages it generated.
- Take a reply message and see what message to which it is in reply.

To trace messages that go through the engine, you must record information each time a message ID changing action happens. Information is recorded every time a message with a new message ID is created, a message ID is reassigned, or a message is stopped or outputted.

If only the message ID and the source message ID are recorded at each of these points, then you cannot trace where each message came from and is going.

After this information is recorded, a command-line interface takes the message ID of an inbound message and returns the message IDs of all outbound messages resulting from it. It takes the message ID of an outbound message and returns the message ID of the message where it ultimately originated.

Inter-site routing requires that a special additional tracing record be recorded on the destination site.

See [Inter-site route](#).

Message tracing levels

This table shows the levels of message tracing control through the GUI:

Level	Description
Site	<p>This is configured in Site Preferences > External Database</p> <p>This requires an engine restart and reopening the Network Configurator.</p> <p>When an external database is configured and message tracing is enabled, the process level and thread level are also enabled. The site level is the main "switch."</p>
Process	<p>This is configured in the Process Configuration dialog box. (Process > Configure).</p> <p>When this level is enabled, this controls all threads in the process. All messages are traced, even if thread level tracing is disabled.</p>
Thread	<p>This is configured in the Network Configurator's Properties tab. This is the lowest priority level. When the process level is disabled and a thread level is enabled, tracing is enabled only on that thread.</p>

Message tracing table schema in external database

This table shows the message tracing table schema:

Name	Type	Description
1 TRACESEQID	INT	NOT NULL

Name	Type	Description
2 HOST1	VARCHAR(256)	NOT NULL
3 SITE1	VARCHAR(256)	NOT NULL
4 TRACEHOSTID	VARCHAR(24)	NOT NULL
5 TRACETIMESTAMP	BIGINT	NOT NULL
6 TRACETYPE	VARCHAR(12)	NOT NULL
7 MID1	VARCHAR(16)	NOT NULL
8 TRACEMID	VARCHAR(16)	NULL
9 ORIGSRCMID	VARCHAR(16)	NULL
10 MSGTYPE	VARCHAR(8)	NULL
11 PROTOCOL	VARCHAR(24)	NULL
12 THREAD	VARCHAR(128)	NULL
13 SMAT1	VARCHAR(256)	NULL
14 TPSPROC	VARCHAR(100)	NULL
15 TPSSTAGE	VARCHAR(24)	NULL
16 TPSDISP	VARCHAR(12)	NULL
17 XLTACTION	VARCHAR(12)	NULL
18 XLTNAME	VARCHAR(100)	NULL
19 TRXID	VARCHAR(100)	NULL
20	VARCHAR(128)	NULL
ROUTEDEST		
21 SOURCEHOST	VARCHAR(256)	NULL
22 SOURCEROOT	VARCHAR(256)	NULL
23 SOURCESITE	VARCHAR(128)	NULL
24 SOURCEDEST	VARCHAR(128)	NULL
25 DESTNAME	VARCHAR(128)	NULL
26 DESTHOST	VARCHAR(256)	NULL
27 DESTSITE	VARCHAR(128)	NULL
28 DESTPROCESS	VARCHAR(128)	NULL
29 DESTTHREAD	VARCHAR(128)	NULL

Name	Type	Description
30 DESTPORT	INT	NULL
31 ERRCONTEXT	VARCHAR(100)	NULL

hcmsgtrace

This command traces messages as far as information is recorded in the tracing database.

```
hcmsgtrace {-l|-L|-w|-d|-v}
```

- `-l` lists messages waiting to be written to external database.
- `-L` list messages with Long format.
- `-w` writes messages to the external database.
- `-d` deletes messages without writing to the external database.
- `-v` verifies the connection to the external database.

Comparing formatted files

When specifying the file path, use the path of the NetConfig formatted file. This is the path on the host that is connected to the IDE.

When you click **Browse**, it defaults to the `revisions` directory of the current site. You can only browse NetConfig formatted files under the current host's root directory.

If any error happens during the script running, such as the file is not found, then the script prompts you with the error information before exiting.

For example, if the NetConfig formatted file that is specified by a parameter cannot be found, then the script prompts you with:

```
Error: could not read "C:/cloverleaf/cisversion/integrator/siteProto/NetConfig": no such file or
directory. This applies to both the command line and IDE mode. If any error happens in IDE mode, then the
error information is shown in the Compare dialog box.
```

Comparing formatted files using the command line

You can compare any two NetConfig formatted files in the file system by specifying file paths as the values of the first and second parameters. In this case, the `setroot` and `setsite` commands also must be invoked before the command line is available.

- 1 `setroot` and `setsite` to the correct site.
- 2 Run the `hcinetdiff.htc` command to compare a given NetConfig formatted file with the current NetConfig formatted file.

The syntax for running `hcinetdiff.htc` is:

```
hcinetdiff baseNetConfig [anotherNetConfig]
```

baseNetConfig is a required parameter that specifies the path to the name of a Network Configurator file.

[*anotherNetConfig*] is an optional parameter that specifies the path to the name of the second Network Configurator file to be compared with *baseNetConfig*. If this parameter is not specified, then the comparison is performed between the current NetConfig formatted file and the file specified in *baseNetConfig*.

Note: The path to the name of the Network Configurator file is relative. That is, the path is relative to the current site in the host server file system. For example, if the current site is `C:\cloverleaf\cis6.2\curr_site, ...\anothersite\NetConfig` represents the `C:\cloverleaf\cis6.2\anothersite\NetConfig` file.

Comparing formatted files using the IDE

- 1 Select **File > Compare** or the toolbar button to compare NetConfig formatted files in the NetConfig view. The **Compare NetConfig Formatted Files** dialog box is displayed. In this dialog box, you can compare one file with the current NetConfig formatted file of the current site, or compare two NetConfig formatted files.

Use this dialog box to compare one file with the current NetConfig formatted file of the current site, or to compare two NetConfig formatted files.

- 2 Specify the file name in the text box, or click **Browse** to navigate to the file location. The file to be compared must be located under the same root. The file browser is for navigating only under the same root.
- 3 Select **Compare with the NetConfig file of the site** or **Compare two NetConfig formatted files**.

When **Compare with the NetConfig file of the site** is selected, **Second NetConfig Formatted File** is disabled. Specify the file name in **First NetConfig Formatted File** or click **Browse** to navigate to the file location. Then, you can select the file to compare with the NetConfig file of the site. This is the default. The current file must be saved before comparing. Otherwise, when **Compare** is clicked, you are reminded that the comparison is based on the previously saved version.

When **Compare two NetConfig formatted files** is selected, **Second NetConfig Formatted File** is enabled. Specify the file names, or click **Browse** to navigate to the file locations.

- 4 Click **Compare**. The results are shown in the **Compare NetConfig Output** dialog box.

The results describe the differences about certain components such as processes and threads in the NetConfig formatted files.

For example, The new site has thread(s) not in the old: conn_1.

If you compare with the NetConfig file of the site, then "new" indicates the current NetConfig formatted file. "old" indicates the NetConfig formatted file specified by *baseNetConfig*.

If a comparison is performed between two NetConfig formatted files, then the result uses terms such as "first NetConfig" and "second NetConfig" instead of "old" and "new." The first file indicates the formatted file specified by *baseNetConfig*. The second file indicates the formatted file specified by [*anotherNetConfig*].

This applies to both command line and dialog box mode.

- 5 Click **Done** to close the dialog box.

Message tracing examples

These examples show various configurations, including:

- Routes and replies
- Reroutes and stops
- intersite routes

Examples are also provided of the information that the tracing database records for each thread.

Routes and replies

In this example, there is a TCP/IP bridge from conn_4 to conn_1 and a message with message ID 0.0.1 comes inbound to conn_1.

A reply is sent back to conn_4 with message ID 0.0.2, which is stopped. Message 0.0.3 goes outbound at conn_2 and 0.0.4 goes through an additional transformation to become 0.0.5 before going outbound at conn_3.

conn_2 is also a TCP/IP bridge that sends to conn_5. conn_5 sends a reply that is received at conn_2 as 0.0.7. it left conn_5 as 0.0.6, but arrives in conn_2 as a new inbound reply 0.0.7. conn_2 does not know that 0.0.7 left conn_5 as 0.0.6. This reply is then routed back to conn_1, where it is stopped as 0.0.8.

Recorded data: routes and replies

In this example, conn_2 also has an inbound SMAT named conn_2_ib and an outbound SMAT named conn_2_ob.

conn_4 and conn_5 do not have tracing enabled. The tracing database has recorded this information:

- MID 0.0.1
 - SRCMID: {}
 - THREAD: conn_1
 - TYPE: data
 - ACTION: ibprotocol
 - DETAIL: tcpip
 - TIME STAMP: 1306270859
- MID 0.0.2
 - SRCMID: 0.0.1
 - THREAD: conn_1
 - TYPE: reply
 - ACTION: tps
 - DETAIL: hl7Raw_ack
 - TIME STAMP: 1306270859
- MID 0.0.2
 - SRCMID: 0.0.1
 - THREAD: conn_1

- TYPE: reply
 - ACTION: obprotocol
 - DETAIL: tcpip
 - TIME STAMP: 1306270859
- MID 0.0.3
 - SRCMID: 0.0.1
 - THREAD: conn_1
 - TYPE: data
 - ACTION: xlated
 - DETAIL: raw
 - TIME STAMP: 1306270859
- MID 0.0.3
 - SRCMID: 0.0.1
 - THREAD: conn_2
 - TYPE: data
 - ACTION: SMAT
 - DETAIL: conn_2_ob
 - TIME STAMP: 1306270859
- MID 0.0.3
 - SRCMID: 0.0.1
 - THREAD: conn_2
 - TYPE: data
 - ACTION: obprotocol
 - DETAIL: tcpip
 - TIME STAMP: 1306270859
- MID 0.0.4
 - SRCMID: 0.0.1
 - THREAD: conn_1
 - TYPE: data
 - ACTION: xlated
 - DETAIL: raw
 - TIME STAMP: 1306270859
- MID 0.0.5
 - SRCMID: 0.0.4
 - THREAD: conn_1
 - TYPE: data
 - ACTION: xlated
 - DETAIL: raw
 - TIME STAMP: 1306270859
- MID 0.0.5
 - SRCMID: 0.0.4
 - THREAD: conn_3
 - TYPE: data
 - ACTION: obprotocol

- `DETAIL: tcpip`
 - `TIME STAMP: 1306270859`
- `MID 0.0.7`
 - `SRCMID: 0.0.3`
 - `THREAD: conn_2`
 - `TYPE: reply`
 - `ACTION: ibprotocol`
 - `DETAIL: tcpip`
 - `TIME STAMP: 1306270859`
- `MID 0.0.7`
 - `SRCMID: 0.0.3`
 - `THREAD: conn_2`
 - `TYPE: reply`
 - `ACTION: SMAT`
 - `DETAIL: conn_2_ib`
 - `TIME STAMP: 1306270859`
- `MID 0.0.8`
 - `SRCMID: 0.0.7`
 - `THREAD: conn_2`
 - `TYPE: reply`
 - `ACTION: xlated`
 - `DETAIL: raw`
 - `TIME STAMP: 1306270859`
- `MID 0.0.8`
 - `SRCMID: 0.0.7`
 - `THREAD: conn_1`
 - `TYPE: reply`
 - `ACTION: stopped`
 - `DETAIL: {}`
 - `TIME STAMP: 1306270859`

Reroutes and stops

In this example, message 0.0.1 comes into conn_1 and is rerouted back to conn_1, resulting in a change of message ID to 0.0.2.

This is routed to conn_2, where it is stopped as 0.0.3, and conn_3, where it is sent through an outbound protocol.

Recorded data: reroutes and stops

In this example, the tracing database has recorded this information:

- MID 0.0.1
 - SRCMID: {}
 - THREAD: conn_1
 - THREAD: data
 - ACTION: ibprotocol
 - DETAIL: tcpip
 - TIME STAMP: 1306270859
- MID 0.0.2
 - SRCMID: 0.0.1
 - THREAD: conn_1
 - THREAD: data
 - ACTION: rerouted
 - DETAIL: {}
 - TIME STAMP: 1306270859
- MID 0.0.3
 - SRCMID: 0.0.2
 - THREAD: conn_1
 - THREAD: data
 - ACTION: xlated
 - DETAIL: raw
 - TIME STAMP: 1306270859
- MID 0.0.3
 - SRCMID: 0.0.2
 - THREAD: conn_2
 - THREAD: data
 - ACTION: stopped
 - DETAIL: {}
 - TIME STAMP: 1306270859
- MID 0.0.4
 - SRCMID: 0.0.2
 - THREAD: conn_1
 - THREAD: data
 - ACTION: xlated
 - DETAIL: raw
 - TIME STAMP: 1306270859
- MID 0.0.4
 - SRCMID: 0.0.2
 - THREAD: conn_3
 - THREAD: data
 - ACTION: obprotocol
 - DETAIL: tcpip
 - TIME STAMP: 1306270859

Inter-site route

In this example, there are two sites. The first one, named site1, has an inbound thread conn_1, routing to an inter-site destination. This is another thread also named conn_1, but in a site named site2.

The inbound thread conn_1 receives a message 0.0.1 from the inbound protocol. It routes to an inter-site destination, which results in the message ID changing to 0.0.2.

In the second site, there is a single outbound thread conn_1 which is the destination pointed to by dest_1 in the first site.

The message it receives through the inter-site route has a message ID of 0.1.1 and a source message ID of 0.0.2. This is the message ID of the message in site1 that was routed here.

Note: This is a message ID in site1, and there might or might not be another message with message ID 0.0.2 in site2.

0.1.1 is sent through the outbound protocol.

Recorded data: inter-site route for site1

The tracing database for site1 has recorded this information:

- MID 0.0.1
 - SRCMID: {}
 - THREAD: conn_1
 - TYPE: data
 - ACTION: ibprotocol
 - DETAIL: tcpip
 - TIME STAMP: 1306270859
- MID 0.0.2
 - SRCMID: 0.0.1
 - THREAD: dest_1:hostname:6666:site2::conn_1
 - TYPE: data
 - ACTION: obintersite
 - DETAIL: {}
 - TIME STAMP: 1306270859

Recorded data: inter-site route for site2

The tracing database for site2 has recorded this information:

- MID 0.1.1
 - SRCMID: 0.0.2
 - THREAD: conn_1
 - TYPE: data
 - ACTION: ibintersite
 - DETAIL: hostname:site1:dest_1

- TIME STAMP: 1306270859
- MID 0.1.1
 - SRCMID: 0.0.2
 - THREAD: conn_1
 - TYPE: data
 - ACTION: obprotocol
 - DETAIL: tcpip
 - TIME STAMP: 1306270859

Details of `ibintersite` records are available because information about the message's inter-site source is stored in the **SOURCECONN** metadata field.

Clearing the tracing database

To manage the tracing database, `hcidbinit` has this option:

```
hcidbinit [-t]
```

`-t` initializes the tracing database.

In addition, the tracing database is also initialized along with the other databases when `hcidbinit -a` is called.

SRCMID

It is important to note that for tracing to work, the definition of the metadata field `SRCMID` must be expanded. In previous versions, the `SRCMID` was only assigned when a message's `MID` was reassigned. In this case, the `SRCMID` was assigned to the message's previous `MID`. The `SRCMID` would then remain null whenever a new message was created, whether in `Xlate`, `TPS`, `reroute`, or any other situation.

Now, whenever a new message is created from a previous message, the new message also has its `SRCMID` assigned to the `MID` of the preceding message. For example, the `SRCMID` of a reply message is the `MID` of the original message that generated the reply. A rerouted message shows up as a new message, but the `SRCMID` is the `MID` of the original message.

A new message that is generated in a translation or procedure also has the `SRCMID` point to the message that was being translated or had the procedure called on.

SOURCECONN and ORIGSOURCECONN

In previous versions, a message that undergoes inter-site routing would arrive at the destination site with `SOURCECONN` and `ORIGSOURCECONN`. These contain the name of the source thread on the source site, but with no indication of the source site.

To provide additional information when tracing backwards across an inter-site route, the metadata SOURCECONN and ORIGSOURCECONN are modified during inter-site routing. This information is included about the source site:

```
srcHost:srcSite:srcProcess:srcThread
```

This format is based on the inter-site DESTCONN format:

```
destName:destHost:destPort:destSite:destProcess:destThread
```

NetConfig

Whether to save tracing information to the tracing database is configurable per thread. NetConfig uses the SAVETRACING sub-key for the SAVEMSGS key:

SAVETRACING specifies if saving of tracing information is enabled for this thread. If this key is "0," or missing, then message tracing is disabled for this thread.

If SAVETRACING is set for a thread, then all traceable actions for messages owned by that thread are recorded in the tracing database. Additionally, a message-level metadata flag is set for the message. This continues to be traced even as it moves through other threads that may not have SAVETRACING set.

If SAVETRACING is not set, then no actions under that thread are recorded unless the individual message's tracing flag is set.

siteInfo

The `messagetracingmaxage=10` key in `siteInfo` is used to prevent the tracing database from growing indefinitely large.

This is the maximum age in days to keep tracing records in the tracing database. The engine automatically purges tracing records earlier than this age every time a process in the site is started, and every 24 hours thereafter.

A value of zero means that there is no maximum age, and the user must clean out the tracing database manually.

The `siteInfo` also contains the `messagetracing=0` key. This enables tracing for the entire site and save the user the trouble of turning it on for every individual thread in the site. "0" indicates "off" and "1" indicates "on." The default to "0."

Tracing first checks this setting, the thread-level setting, and finally the message-level flag to determine if a message is to be traced.

Leveraging CIS in a service-oriented architecture

Most healthcare application vendors support the HL7 2.x.x standards in a variant format using TCP/IP MLLP protocol for communications. New vendors in the healthcare space are constructing their applications to use web services in a service oriented architecture.

This describes the methods used to create Cloverleaf threads that accept a defined SOAP service. This service returns data back to the calling client in a predefined format.

The code and example are available in BOX format using CIS 6.1 and later versions.

The inbound and outbound data are base64 encoded.

Configuration steps:

- 1 [Cloverleaf setup and defining the XSD](#)
- 2 [Saving the xsd file](#)
- 3 [Configuring the XSD WSDL](#)
- 4 [Creating the HTTP inbound thread](#)
- 5 [Creating the HTTPS inbound thread](#)
- 6 [Starting the wstest process](#)
- 7 [Creating a Cloverleaf BOX](#)

The [Examples](#) use HTTP and HTTPS, take the Cloverleaf site name, translation, and HL7 V2 data and return the translated HL7 data.

Requirements for configuring the CAA-WS adapter as a web service

- Cloverleaf 6.1 or later
- CAA-WS 2.0
- Knowledge of XML, SOAP, and WSDL
- Basic understanding of network security

Helpful open source tools

- soapUI: <http://www.soapui.org>
- protecle: <http://sourceforge.net/projects/portecle/>
- Oxygen XML: <http://www.oxygenxml.com/>
- XML Spy: Not open source

Cloverleaf setup and defining the XSD

Cloverleaf setup consists of configuring the CAA-WS adapter as a web service.

Using an XML editor, create an XML Schema Definition XSD. In this example, `clInbound` and `clOutbound` are defined.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) -->
<xs:schema
```

```

targetNamespace="http://www.infor.com/clXLT"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.infor.com/clXLT"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:element name="clInbound">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="clSite" type="xs:string" minOccurs="0" maxOccurs="1" />
      <xs:element name="xltName" type="xs:string"/>
      <xs:element name="dataIn" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="clOutbound">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="dataOut" type="xs:string" minOccurs="0" maxOccurs="1" />
      <xs:element name="errorOut" type="xs:string" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>

```

Saving the xsd file

- 1 Save the `xsd` file in the Cloverleaf site XML package directory. This lets Cloverleaf compile the schema for use in translations.
- 2 In the XML Package Manager, create a new package folder and place the `xsd` file in the folder.
- 3 Select the `xsd` file and compile using the default values. In the **Compile Options** dialog box, only **Use the Default Namespace in XSD File** is selected.

Configuring the XSD WSDL

The WSDL defines all inbound and outbound structures and the communication methods that are used in the process.

The Cloverleaf CAA-WS adapter provides tools for automatically generating the WSDL. You must log in to the Cloverleaf server to launch the XSD WSDL tool.

- 1 Log in to the Cloverleaf server using the `hci` user. Run the `setsite` command, for example, `setsite alpha`.
- 2 Change to the `$HCIR00T/CAA/ws/tool/xsdWsdLTool` directory and open the XSD WSDL Tool application.
- 3 Select the XML package, `xsd` file, and the inbound and outbound elements.
- 4 Select the SOAP version to use and click **Compile** to generate. This compiles the schema with the additional SOAP headers in the Cloverleaf format.
- 5 In the WSDL section update the **Service URL** field to indicate how the service is to be exposed. For example: `http://hostname:8080/newXLT`.
- 6 Click **Generate WSDL**. This creates the WSDL that you can share with any organization using this service.

Creating the HTTP inbound thread

- 1 In Network Configurator, create a Cloverleaf thread and select the java/ws-server protocol.
- 2 Click **Properties** to open the **WS Server** dialog box.
- 3 Click **New SOAP Server** and specify the **WSDL URI** that was created in the Generate WSDL process.
- 4 Clear the address section to use the definition and specify `http://hostname:8081/c1xlt`. The default service mode is "payload," which is used in this example.
- 5 Click **OK**.

Creating the HTTPS inbound thread

- 1 In Network Configurator, create a Cloverleaf thread and select the java/ws-server protocol.
- 2 Click **Properties** to open the **WS Server** dialog box.
- 3 Click **New** and select **Soap Provider**.
- 4 Specify the **WSDL URI** that was created in the Generate WSDL process.
- 5 Clear the address section to use the definition and specify `https://hostname:8080/c1xlt`. The default service mode is "payload," which is used in this example.
- 6 Click **OK**.

Note: The inbound thread can use both HTTP and HTTPS, as long as other ports are assigned. This example uses port 8080.
- 7 On the **WS Server** dialog box, click **New** and select **Engine** to create the Jetty engine instance.
- 8 Specify the **Port** and **Host**. This example uses *hostname*, so **Host** can be blank.
- 9 Click **OK**.
- 10 To configure the policy properties, select **TLS secured**.
- 11 For **Secure Socket Protocol**, select **TLSv1**.
- 12 Specify the **Key Password**, **Keystore File**, and **Keystore Password**.
- 13 Because this example does not use client authentication, select **false** for **Client Authentication Required** and **Client Authentication Wanted**.
- 14 Click **OK**.

Examples

These examples use HTTP and HTTPS, take the Cloverleaf site name, translation, and HL7 V2 data and return the translated HL7 data.

The `soapUI` interface is used as a sending service (client) to connect to the Cloverleaf server providing the service.

HL7 v2 contains characters such as "&" that could be interpreted incorrectly by HTML (&), therefore base64 encoding of the message payload is used. This method has been accepted by the healthcare industry as the standard for transmitting HL7 V2 data in formatted XML and IHE XDS.b implementations.

Inbound data before translate

```
MSH|^~/&|MISYS|MISYS|VENDOR|VENDOR|20061212062029||ADT^A08|5255|P|2.3|||AL|
EVN||20061212062029|||2|
PID||634568|27356^^^MISYS^PI~634568^^^MISYS^PT~123-45-5321^^^USSA^SS|634568|SAM
PLE^MISYS^E||19780621|F|||2 WEST STREET^^CITY^MN^22118|||(761
)561-3411|||1^SINGLE^MISYS^^SINGLE^VENDOR||634568|123-45-5321|
ZPT|1^A-F^MISYS|0|20001207||
ZPV|LITTLE MISYS/SISTER|152 413 1535|STUDENT||1^SINGLE^MISYS|100^SCANNED CHART^MISYS|27^FAMI
LY/FRIEND^MISYS||
NK1|1|||||PTDEM|||||||0|
PV1||I|^A^2|R||1^^^MISYS&1^^^^^AWH SD|7^NADE
MD^JOHN^C||PX|||||188666^^^MISYS|7|||||
ZP1|G27165|38275|BCBS|||||T|ANNUAL EXAM|14416503|||||
PV2|||||200612131440||||ANNUAL EXAM|
ROL|LI|AT^ATTENDING^MISYS|7^NADE MD^JOHN^C||
ROL|LI|LOC^PRIOR PATIENT LOCATION^MISYS|1^AWH SD|||||67 DREW ST S^^CITY^VA^12345|
DG1|1|I9|V72.31^EXAM GYNECOLOGICAL^MISYS^V72.31^EXAM GYN^MISYS|EXAM GYN|||||7^NADE
MD^JOHN^C^^^^MISYS|
ZDG||
GT1|1|27356^^^MISYS^GI~634568^^^MISYS^GN|SAMPLE^MISYS^E||24 RITA LANE^^CITY^MN^55318|(761)561-
3411||||^9999^UNKNOWN^VENDOR|||||
|||||
IN1|1|4^^^MISYS|BCBS^^^^^MISYS|PO BOX 638^^CITY^VA^12345|||(151)612-5100^^PH|EC013W1|||||SAM
PLE^MISYS^E|2^02-FEMALE SUBSCRIBER^MISYS^|197
80621|2 WEST STREET^^CITY^MN^22118|Y|Y|||||XZAXZ3511186|||||634568^^^MISYS^PT|
IN2||123-45-5321||I|||||XZA
XZ3511186|||||2^02-FEMALE SUBSCRIBER^MISYS^^02-FEMALE S
UBSCRIBER^VENDOR^|
ZIN|47502|0|B|20030421|27356^^^MISYS^PI|||1|
```

Outbound data after translate

```
MSH|^~/&|MISYS|MISYS|VENDOR|VENDOR|20061212062029||ADT^A08|5255|P|2.3|||AL|
EVN||20061212062029|||2|
PID||634568|27356^^^MISYS^PI~634568^^^MISYS^PT~123-45-5321^^^USSA|634568|SAM
PLE^MISYS^E||19780621|F|||2 WEST STREET^^CITY^MN^22118|||(761)561-3411|||1||634568|123-45-5321
NK1|1|||||PTDEM|||||||0|
PV1||I|^A^2|R||1^^^MISYS&1^^^^^AWH SD|7^NADE MD^JOHN^C||PX|||||188666^^^MISYS|7
PV2|||||20061213||||ANNUAL EXAM
DG1|1|I9|V72.31^EXAM GYNECOLOGICAL^MISYS^V72.31^EXAM GYN^MISYS|EXAM GYN|||||7^NADE
MD^JOHN^C^^^^MISYS
GT1|1|27356^^^MISYS^GI~634568^^^MISYS^GN|SAMPLE^MISYS^E||24 RITA LANE^^CITY^MN^55318|(761)561-
3411
IN1|1|4^^^MISYS|BCBS^^^^^MISYS|PO BOX 638^^CITY^VA^12345|||(151)612-5100^^PH|EC013W1|||||SAM
PLE^MISYS^E|2|19780621|2 WEST STREET^^CITY^MN^22118|Y|Y|||||XZA
XZ3511186|||||634568^^^MIS
IN2||123-45-5321||I|||||XZA
XZ3511186|||||2^02-FEMALE SUBSCRIBER^MISYS^^02-FEMALE SUBSCRIBER^VENDOR
```

Starting the wstest process

Open the Network Monitor and start the inbound Cloverleaf process wstest.

Creating a Cloverleaf BOX

Create a BOX that includes:

- Threads wstest_in and wstest_out.
- Tclproc processCLXLT.

- Translate example `rfa_adt.xlt`.
- Test data files `example_A08_in.xml` and `our_adt_out.dat`.

Outbound tab

This table shows the configurable parameters on the **Outbound** tab:

Parameter	Description
Hold	This holds any outbound messages, depending on thread type, including ACKs.
TPS Outbound Data	Click Edit to open the TPS Editor . Then, click Add to open the TPS Properties dialog box. Select the procedures or Java class and arguments for outbound data messages. This TPS is used to post-process the outbound data messages after translation but before being written to the connection.
Retries	<p>This is the maximum number of times to attempt transmission of an outbound message to the destination thread. A retry happens when a protocol write fails.</p> <ul style="list-style-type: none">• 0 is the initial transmission only.• 1 is the initial transmission plus one retry.• -1 is unlimited retries.
Interval	This is the number of seconds to wait between each retry.
Send OK Procs	Click Edit to open the TPS Editor . Then, click Add to open the TPS Properties dialog box. Select the procedures or Java class and arguments to run upon successful transmission of the outbound message to the destination thread.
Send Fail Procs	Click Edit to open the TPS Editor . Then, click Add to open the TPS Properties dialog box. Select the procedures or Java class and arguments to run upon unsuccessful transmission of the outbound message to the destination thread. An unsuccessful transmission happens when the number of retries fails. All retries and forwarding threads are exhausted before a transmission is treated as failed.

Parameter	Description
Prewrite Procs	<p>Click Edit to open the TPS Editor. Then, click Add to open the TPS Properties dialog box, from where you can select the procedures or Java class and arguments. These procedures provide user-control immediately before the protocol write happens.</p> <p>The only dispositions permitted in the prewrite are CONTINUE and KILL.</p>
Forward Thread	<p>Click List to open the Threads list box. Use the list box to select the thread to serve as an alternate destination for the outbound messages. The engine first exhausts the permissible number of transmission retries to the primary destination thread before forwarding the message to the alternate thread.</p>
EDI Batch	<p>Click the arrow to select the format from the list. Then, click Edit to open the batch configuration dialog box. The creation of a batch is triggered by a Message Count or an Interval.</p> <p>This feature requires an additional license to be functional.</p>

Inbound Replies pane

This table shows the configurable inbound reply parameters:

Parameter	Description
Await Replies	<p>When this is enabled, the engine generates a reply message based on the returned results.</p> <p>Clearing this disables all other configuration entries in this pane.</p> <p>If a multi-server is configured to await replies, then for each message sent the thread enters Await Replies mode. Only the destination client of the last outbound message is read for a reply message.</p> <p>Await Replies is a thread-level operation. That is, there is not a separate await reply for each client. Awaiting replies blocks all traffic through the thread until the target client responds or a time-out happens.</p> <p>OBMSGID is available when Await Replies is selected.</p> <p>See DICOM SCU Await Replies handling on page 523 for further information.</p>

Parameter	Description
Timeout	<p>This is the time-out limit. The default is 45. The default value for this feature can be changed on the Netconfig/NetMonitor tab of the Client Preferences dialog box. Clicking Advanced opens the Configure Default Values dialog box. Having a time-out of 45 and using <code>hcitpsmsgkill</code> helps to avoid unexpected transmission</p>
Timeout Handling	<p>This opens the Timeout Handling dialog box.</p> <p>Select Resend OB message to automatically resend the message on time-out until a reply is received. This is the default.</p> <p>Select Reply Generation to select from three types of replies:</p> <ul style="list-style-type: none"> • None is for no reply generation. • Text is for text reply generation. Click Edit to open the Tcl Script Editor. Use this dialog box to edit existing files or procs. • Procs is for Tcl reply generation. Click Edit to open the TPS Editor. Then, click Add to open the TPS Properties dialog box, from where you can select the procedures or Java class and arguments. <div style="background-color: #ffffcc; padding: 10px; margin: 10px 0;"> <p>Caution: Be careful when using reply handling. If the receiver does not detect and handle duplicate messages, then this could potentially lead to unnecessary extra messages displaying on the receiving end.</p> </div> <p>Do not use <code>KILL</code> to stop inbound reply messages. <code>KILL</code> causes the Await Reply mode to shut down abnormally, sending you to time-out handling every time. Use <code>KILLREPLY</code> instead. If the reply is <code>CONTINUE</code>d or ignored, then the engine automatically shuts down Await Reply mode. The inbound reply messages should never be stopped using <code>KILL</code>.</p>

Parameter	Description
TPS Inbound Reply	<p>Click Edit to open the TPS Editor. Then, click Add to open the TPS Properties dialog box, where you can select the procedures or Java class and arguments for the inbound reply messages. This TPS is used to post-process the inbound reply messages after translation but before being written to the connection.</p> <p>The default is <code>hcitpsmsgkill</code>. The default value for this feature can be changed on the Netconfig/Net-Monitor tab of the Client Preferences dialog box. Clicking Advanced opens the Configure Default Values dialog box.</p>
Trx ID Determination Format	<p>Click the arrow to select the format from the list. Then, click Edit to open the Trx ID determination dialog box. Use this dialog box to configure transaction ID information:</p> <ul style="list-style-type: none"> For FRL, select the offset and length. For HL7, HPRIM, X12, NCPDP, and UN/EDIFACT, it is not necessary to select the version and variant. Edit is not available when any of these are selected. Selecting XML disables Edit. There are no properties to edit. For XML (Namespace), selecting XML (Namespace) disables Edit. There are no properties to edit. Select UPoC to specify a UPoC without going through FRL. Click Edit to open the Format UPoC dialog box. Only one proc can be selected. Specify arguments for inbound data messages.

DICOM SCU Await Replies handling

When a DICOM SCU thread is created, there is always a response message from the SCP side. The internal DICOM response message is treated as the reply message.

When **Await Replies** is enabled, a `KILLREPLY` disposition for the internal reply message handle is required and a new reply message must be configured.

To do this:

- 1 Navigate to **Network Configurator > Outbound tab > Inbound Replies pane**.
- 2 Under Inbound Replies, for **TPS Inbound Reply**, select "ib reply" and click **Edit**.

3 Add these lines:

```
#####
# Name:      id_reply
# Purpose:   <description>
# UPoC type: tps
#####

proc ib_reply { args } {
    global HciConnName
    keylget args MODE mode
    set ctx "" ; keylget args CONTEXT ctx
    set module "id_reply/$HciConnName/$ctx"
    set uargs {} ; keylget args ARGS uargs
    set debug 0
    catch {keylget uargs DEBUG debug}
    set dispList {}

    switch -exact -- $mode {
        start {}

        run {
            keylget args MSGID mh
            set msg [msgget $mh]

            echo RECEIVED DICOM REPLY

            set newreply [msgcreate -recover -type reply [msgget $mh]]
            set resp_driverctl [msgmetaget $mh DRIVERCTL]
            msgmetaset $newreply DRIVERCTL $resp_driverctl

            lappend dispList "KILLREPLY $mh" "CONTINUE $newreply"
        }

        time {}
        shutdown {}
        default {
            error "Unknown mode '$mode' in $module"
        }
    }

    return $dispList
}
```

Note: The KILLREPLY disposition is for the internal reply message handle, and the CONTINUE disposition is for the new reply message.

Reply generation

This is located in the **Timeout Handling** dialog box. It is called only if the engine is expecting an inbound protocol reply message, but does not receive one. This UPoC exists because replies must be received within a configurable time-out period.

The input **msgID** is a copy of the outbound data message that generated no reply. It is meant to be used as a reference for generating the inbound reply.

Caution: If the input **msgID** is returned with **PROTO** disposition, then the message is placed directly on the OB Processing queue. This message is not recoverable. To resend a recoverable message, the **OBMSGID** should be used.

The **OBMSGID** is the handle for the saved outbound data message that generated no reply.

Typically, use reply generation to:

- Resend the outbound data message.
- Receive zero or more message handles that are used as inbound reply protocol messages.

A source protocol thread waits on a reply for a defined amount of time. If no time-out is configured, then it waits indefinitely.

EDI batch for X12

An X12 Batch is the interchange of one or more encoded business transactions including the EDI (electronic data interchange) encoded transactions of Accredited Standards Committee (ASC) X12. The batch message provides the interchange envelope of a header and trailer for the electronic interchange through a data transmission.

A batch message is defined in `$HCIR00T/formats/x12/<version>/batch`.

The basic batch message hierarchy is similar to:

```
ISA
[ISB]
[ISE]
[{TA1}]
{
  GS
  [S1S]
  ... (Transaction set Message)
  [{
    S1A
    SVA
  }]
  [S1E]
  GE
}
```

Header and trailer definitions:

Segment	Definition
GS	Functional Group Header. This starts a group of related transaction sets.
ST	Transaction Set Header. This starts a transaction set.
LS	Loop Header. This starts a bounded loop of data segments but is not part of the loop.
LE	Loop Trailer. This ends a bounded loop of data segments but is not part of the loop.
SE	Transaction Set Trailer. This ends a transaction set.
GE	Functional Group Trailer. This ends a group of related transaction sets.

More than one ST/SE pair, each representing a transaction set, can be used within one functional group. More than one LS/LE pair, each representing a bounded loop, can be used within one transaction set.

According to this structure, you can define to split to groups/transaction sets during the handling of an Inbound Batch message.

Cloverleaf splits the batch message by searching GS/ST segment, then puts the whole group/transaction set content as a single X12 message.

On the outbound side, the engine puts several single X12 messages together by adding the envelope header, ISA – Interchange Control Header, and trailer, IEA - Interchange Control Trailer. The user must set the values for these segments.

Examples

A dialog box opens when you click **Edit** for **EDI Batch** on the **Outbound** tab.

On the Inbound side, set the Inbound **EDI Batch** to X12, and **Trx ID Determination Format** to X12. When a batch message is received, the Cloverleaf engine splits the batch message according to the setting (Split to Transaction Sets/Groups) to several single messages. Then, Cloverleaf gets the TRXID from these messages, and routes them to the destinations.

On the Outbound side, set the Inbound **EDI Batch** to X12 with the default values for the ISA segment and the message count/interval for the batch message. When there are enough messages to be sent out, or the time interval passes, the engine puts these messages to be a batch message by adding ISA and IEA segments.

X12 batch standard

You can select a batch type for inbound and outbound EDI batch type.

Inbound

Inbound X12 batches are supported with an EDI Batch attribute for inbound threads. If the X12 EDI Batch type is specified for an inbound thread, then all inbound messages are treated as batches.

Version options for the GS segment date file length:

- "3XXX" indicates the length is 6 bytes.
- "4XXX" indicates the length is 8 bytes.

Split to options:

- "Groups" option: Each inbound message is split into individual groups for processing. Values from the batch header and trailer are attached to the group message metadata. For example:

```
msgUserData : {ISA {ISA 00 { } 01 {PASSWORD } ZZ {SUBMITTERS.ID } ZZ {RECEIVERS.ID } 020711 1253 U 00401 000000905 1 T :}}
```
- "Transaction Sets": Each inbound message is split into individual transactions for processing. Values from the batch header and trailer are attached to the individual transaction metadata. For example:

```
msgUserData: {ISA {ISA 00 { } 01 {PASSWORD } ZZ {SUBMITTERS.ID } ZZ {RECEIVERS.ID } 020711 1253 U 00401 000000905 1 T :}} {GS {GS HB {SENDER CODE} {RECEIVER CODE} 20020711 1102 2 X 004010X092}}
```

Outbound batch

The outbound batch has an edit screen with these parameters. These are configured from the **Outbound** tab of the Network Configurator **Thread Configuration** dialog box.

Version options for the GS segment data field length:

- "3XXX" indicates the length is 6 bytes.
- "4XXX" indicates the length is 8 bytes.

Header options

The creation of a batch is triggered by a Message Count or an Interval. All of the header fields are required in the final message. You can leave them blank on the screen, as they are set and overridden with DRIVERCONTROL in each message.

This table shows the Header options on the **Outbound** tab of the Network Configurator **Thread Configuration** dialog box:

Option	Description
Authorization Information Qualifier	<p>Code identifying the type of information in the Authorization Information.</p> <p>00: No Authorization Information Present. No Meaningful Information in I02.</p> <p>01: UCS Communications ID</p> <p>02: EDX Communications ID</p> <p>03: Additional Data Identification</p> <p>04: Rail Communications ID</p> <p>05: Department of Defense (DoD) Communication Identifier</p> <p>06: United States Federal Government Communication Identifier</p>
Authorization Information	<p>Information used for additional identification or authorization of the interchange.</p> <p>The sender or the data in the interchange. The type of information is set by the Authorization Information Qualifier.</p> <p>This is alphanumeric text up to 10 characters.</p>
Security Information Qualifier	<p>Code identifying the type of information in the Security Information.</p> <p>00: No Security Information Present. No Meaningful Information in I04.</p> <p>01: Password</p>

Option	Description
Security Information	<p>This is used for identifying the security information about the interchange sender or the data in the interchange.</p> <p>The type of information is set by the Security Information Qualifier.</p> <p>This is alphanumeric text up to 10 characters.</p>
Sender ID/Sender ID Qualifier	See Sender ID Qualifier on page 529.
Receiver ID/Receiver ID Qualifier	See Receiver ID Qualifier on page 531.
Standards Identifier	Code to identify the Agency responsible for the control standard used by the message that is enclosed by the interchange header and trailer.
Version Number	See Version Number on page 541.
Acknowledgment Requested	<p>Code indicating sender's request for an interchange acknowledgment.</p> <p>0: No Interchange Acknowledgment Requested</p> <p>1: Interchange Acknowledgment Requested (TA1)</p>
Usage Indicator	<p>Code indicating whether data enclosed by this interchange envelope is test, production or information.</p> <p>I: Information</p> <p>P: Production Data</p> <p>T: Test Data</p>
Functional ID Code	See Functional ID Code on page 532.
Application Sender's Code	Code identifying party sending transmission. Codes agreed to by trading partners. The maximum length of this string is 15, and the minimum is 2.
Application Receiver's Code	Code identifying party receiving transmission. Codes agreed to by trading partners. The maximum length of this string is 15, and the minimum is 2.
Group Control Number	Assigned number originated and maintained by the sender. The maximum length of this number is 9 and the minimum is 1.
Agency Code	<p>Code identifying the issuer of the standard. This code is used in conjunction with Data Element 480.</p> <p>T: Transportation Data Coordinating Committee (TDCC)</p> <p>X: Accredited Standards Committee X12</p>

Option	Description
Version/Release/Industry Identifier Code	Code indicating the version, release, sub release, and industry identifier of the EDI. Standard being used, including the GS and GE segments.

Sender ID Qualifier

Sender ID Qualifiers are identification codes published by the sender for other parties to use as the receiver ID to route data to them. The sender always codes this value in the sender ID element. The code indicates the system/method of code structure used to designate the sender ID element being qualified. This is alphanumeric text up to 15 characters.

Sender ID numeric qualifiers:

01-09	01: Duns (Dun & Bradstreet) 02: SCAC (Standard Carrier Alpha Code) 03: FMC (Federal Maritime Commission) 04: IATA (International Air Transport Association) 07: Global Location Number (GLN) A globally unique 13 digit code for the identification of a legal, functional, or physical location within the Uniform Code Council (UCC) and International Article Number Association (EAN) numbering system. 08: UCC EDI Communications ID (Comm ID) 09 X.121 (CCITT)
-------	---

10-19	<p>10: Department of Defense (DoD) Activity Address Code</p> <p>11: DEA (Drug Enforcement Administration)</p> <p>12 Phone (Telephone Companies)</p> <p>13: UCS Code (The UCS Code is a Code Used for UCS Transmissions; it includes the Area Code and Telephone Number of a Modem; it Does Not Include Punctuation, Blanks or Access Code).</p> <p>14: Duns Plus Suffix</p> <p>15: Petroleum Accountants Society of Canada Company Code</p> <p>16: Duns Number With 4-Character Suffix</p> <p>17: American Bankers Association (ABA) Transit Routing Number (Including Check Digit, 9 Digit)</p> <p>18: Association of American Railroads (AAR) Standard Distribution Code</p> <p>19: EDI Council of Australia (EDICA) Communications ID Number (COMM ID)</p>
20-29	<p>20: Health Industry Number (HIN)</p> <p>21: Integrated Postsecondary Education Data System, or (IPEDS)</p> <p>22: Federal Interagency Commission on Education, or FICE</p> <p>23: National Center for Education Statistics Common Core of Data 12-Digit Number for Pre-K-Grade 12 Institutes, or NCES</p> <p>24; The College Board's Admission Testing Program 4-Digit Code of Postsecondary Institutes, or ATP</p> <p>25: ACT, Inc. 4-Digit Code of Postsecondary Institutions.</p> <p>26: Statistics of Canada List of Postsecondary Institutions</p> <p>27: Carrier Identification Number as assigned by Health Care Financing Administration (HCFA)</p> <p>28: Fiscal Intermediary Identification Number as assigned by Health Care Financing Administration (HCFA)</p> <p>29: Medicare Provider and Supplier Identification Number as assigned by Health Care Financing Administration (HCFA)</p>

30-38

- 30: U.S. Federal Tax Identification Number
- 31: Jurisdiction Identification Number Plus 4 as assigned by the International Association of Industrial Accident Boards and Commissions (IAIABC)
- 32: U.S. Federal Employer Identification Number (FEIN)
- 33: National Association of Insurance Commissioners Company Code (NAIC)
- 34: Medicaid Provider and Supplier Identification Number as assigned by individual State Medicaid Agencies in conjunction with Health Care Financing Administration (HCFA)
- 35: Statistics Canada Canadian College Student Information System Institution Codes
- 36: Statistics Canada University Student Information System Institution Codes
- 37: Society of Property Information Compilers and Analysts
- 38: The College Board and ACT, Inc. 6-Digit Code List of Secondary Institutions

Sender ID alpha qualifiers:

AM-ZZ

- AM: Association Mexicana delCodigo de Producto (AMECOP) Communication ID
- NR: National Retail Merchants Association (NRMA) - Assigned
- SA: User Identification Number as assigned by the Safety and Fitness Electronic Records (SAFER) System
- SN: Standard Address Number
- ZZ :Mutually Defined

Receiver ID Qualifier

Receiver ID Qualifiers are code indicating the system/method of code structure used to designate the receiver ID element being qualified.

Option	Description
Receiver ID	Identification code published by the receiver of the data; When sending, it is used by the sender as their sending ID. Other parties sending to them use this as a receiving ID to route data to them. Alphanumeric text up to 15 characters.

Option	Description
Receiver ID Qualifier	This is the same as Sender ID Qualifier. See Sender ID Qualifier on page 529.

Functional ID Code

The Functional ID Code identifies a group of application-related transaction sets. These transaction sets are listed alphabetically.

Group	Functional ID Code (Table)	Notes
A	AA: Account Analysis (822)	
	AB: Logistics Service Request (219)	
	AC: Associated Data (102)	
	AD: Individual Life, Annuity and Disability Application (267)	
	AE: Premium Audit Request and Return (187)	
	AF: Application for Admission to Educational Institutions (189)	
	AG: Application Advice (824)	
	AH: Logistics Service Response (220)	
	AI: Automotive Inspection Detail (928)	
	AK: Student Educational Record (Transcript) Acknowledgment (131)	
	AL: Set Cancellation (998)	
	AM: Item Information Request (893)	
	AN: Return Merchandise Authorization and Notification (180)	
	AO: Income or Asset Offset (521)	
	AP: Abandoned Property Filings (103)	
	AQ: U.S. Customs Manifest (309)	
	AR: Warehouse Stock Transfer Shipment Advice (943)	
	AS: Transportation Appointment Schedule Information (163)	
	AT: Animal Toxicological Data (249)	
	AU: U.S. Customs Status Information (350)	
	AV: U.S. Customs Carrier General Order Status (352)	
	AW: Warehouse Inventory Adjustment Advice (947)	
	AX: U.S. Customs Events Advisory Details (353)	
	AY: U.S. Customs Automated Manifest Archive Status (354)	
	AZ: U.S. Customs Acceptance/Rejection (355)	

Group	Functional ID Code (Table)	Notes
B	BA: U.S. Customs Permit to Transfer Request (356)	
	BB: U.S. Customs In-Bond Information (357)	
	BC: Business Credit Report (155)	
	BD: U.S. Customs Consist Information (358)	
	BE: Benefit Enrollment and Maintenance (834)	
	BF: Business Entity Filings (105)	
	BL: Motor Carrier Bill of Lading (211)	
	BS: Shipment and Billing Notice (857)	
C	CA: Purchase Order Change Acknowledgment/Request - Seller Initiated (865)	
	CB: Unemployment Insurance Tax Claim or Charge Information (153)	
	CC: Clauses and Provisions (504)	
	CD: Credit/Debit Adjustment (812)	
	CE: Cartage Work Assignment (222)	
	CF: Corporate Financial Adjustment Information (844 and 849)	
	CH: Car Handling Information (420)	
	CI: Consolidated Service Invoice/Statement (811)	
	CJ: Manufacturer Coupon Family Code Structure (877)	
	CK: Manufacturer Coupon Redemption Detail (881)	
	CL: Election Campaign and Lobbyist Reporting (113)	
	CM: Component Parts Content (871)	
	CN: Coupon Notification (887)	
	CO: Cooperative Advertising Agreements (290)	
	CP: Electronic Proposal Information (251, 805)	
	CQ: Commodity Movement Services Response (874)	
	CR: Rail Carhire Settlements (414)	
	CS: Cryptographic Service Message (815)	
	CT: Application Control Totals (831)	
	CU: Commodity Movement Services (873)	
	CV: Commercial Vehicle Safety and Credentials Information Exchange (285)	
	CW: Educational Institution Record (133)	

Group	Functional ID Code (Table)	Notes
D	D3: Contract Completion Status (567)	D3: Transaction set for reporting the administrative closure status of physically completed contracts.
	D4: Contract Abstract (561)	
	D5: Contract Payment Management Report (568)	
	DA: Debit Authorization (828)	
	DD: Shipment Delivery Discrepancy Information (854)	
	DF: Market Development Fund Allocation (883)	
	DI: Dealer Information (128)	
	DM: Equipment Order (422)	
	DS: Data Status Tracking (242)	
E	DX: Direct Exchange Delivery and Return Information (894, 895)	
	EC: Educational Course Inventory (188)	
	ED: Student Educational Record (Transcript) (130)	
	EI: Railroad Equipment Inquiry or Advice (456)	
	EN: Equipment Inspection	
	EP: Environmental Compliance Reporting (179)	
	ER: Revenue Receipts Statement (170)	
	ES: Notice of Employment Status (540)	
	EV: Railroad Event Report (451)	
F	EX: Excavation Communication (620)	
	FA: Functional or Implementation Acknowledgment Transaction Sets (997, 999)	
	FB: Freight Invoice (859)	
	FC: Court and Law Enforcement Information (175, 176)	
	FG: Motor Carrier Loading and Route Guide (217)	
	FR: Financial Reporting (821, 827)	
G	FT: File Transfer (996)	
	GC: Damage Claim Transaction Sets (920, 924, 925, 926)	
	GE: General Request, Response or Confirmation (814)	
	GF: Response to a Load Tender (990)	
	GL: Intermodal Group Loading Plan (715)	
	GP: Grocery Products Invoice (880)	
	GR: Statistical Government Information (152)	
	GT: Grant or Assistance Application (194)	

Group	Functional ID Code (Table)	Notes
H	HB: Eligibility, Coverage or Benefit Information (271) HC: Health Care Claim (837) HI: Health Care Services Review Information (278) HN: Health Care Information Status Notification (277) HP: Health Care Claim Payment/Advice (835) HR: Health Care Claim Status Request (276) HS: Eligibility, Coverage or Benefit Inquiry (270) HU: Human Resource Information (132) HV: Health Care Benefit Coordination Verification (269)	
I	IA: Air Freight Details and Invoice (110, 980) IB: Inventory Inquiry/Advice (846) IC: Rail Advance Interchange Consist (418) ID: Insurance/Annuity Application Status (273) IE: Insurance Producer Administration (252) IF: Individual Insurance Policy and Client Information (111) IG: Direct Store Delivery Summary Information (882) IH: Commercial Vehicle Safety Reports (284) IJ: Report of Injury, Illness or Incident (148) IM: Motor Carrier Freight Details and Invoice (210, 980) IN: Invoice Information (810) IO: Ocean Shipment Billing Details (310, 312, 980) IR: Rail Carrier Freight Details and Invoice (410, 980) IS: Estimated Time of Arrival and Car Scheduling (421)	
J	JB: Joint Interest Billing and Operating Expense Statement (819)	
K	KM: Commercial Vehicle Credentials (286)	
L	LA: Federal Communications Commission (FCC) License Application (195) LB: Lockbox (823) LI: Locomotive Information (436) LN: Property and Casualty Loss Notification (272) LR: Logistics Reassignment (536) LS: Asset Schedule (851) LT: Student Loan Transfer and Status Verification (144)	

Group	Functional ID Code (Table)	Notes
M	MA: Motor Carrier Summary Freight Bill Manifest (224)	MD: Transactions related to the interchange of information involving material in the supply, control, and distribution systems and financial management of the Department of Defense or other participating agencies.
	MC: Request for Motor Carrier Rate Proposal (107)	
	MD: Department of Defense Inventory Management (527)	
	ME: Mortgage Origination (198, 200, 201, 245, 261, 262, 263, 833, 872)	
	MF: Market Development Fund Settlement (884)	
	MG: Mortgage Servicing Transaction Sets (203, 206, 259, 260, 264, 266)	
	MH: Motor Carrier Rate Proposal (106)	
	MI: Motor Carrier Shipment Status Inquiry (213)	
	MJ: Secondary Mortgage Market Loan Delivery (202)	
	MK: Response to a Motor Carrier Rate Proposal (108)	
	MM: Medical Event Reporting (500)	
	MN: Mortgage Note (205)	
	MO: Maintenance Service Order (650)	
	MP: Motion Picture Booking Confirmation (159)	
	MQ: Consolidators Freight Bill and Invoice (223)	
	MR: Multilevel Railcar Load Details (125)	
	MS: Material Safety Data Sheet (848)	
	MT: Electronic Form Structure (868)	
	MV: Material Obligation Validation (517)	
	MW: Rail Waybill Response (427)	
	MX: Material Claim (847)	
	MY: Response to a Cartage Work Assignment (225)	
	MZ: Motor Carrier Package Status (240)	
N	NC: Nonconformance Report (842)	
	NL: Name and Address Lists (101)	
	NP: Notice of Power of Attorney (157)	
	NR: Secured Receipt or Acknowledgment (993)	
	NT: Notice of Tax Adjustment or Assessment (149)	
O	OC: Cargo Insurance Advice of Shipment (362)	
	OG: Order Group - Grocery (875, 876)	
	OR: Organizational Relationships (816)	
	OW: Warehouse Shipping Order (940)	

Group	Functional ID Code (Table)	Notes
P	PA: Price Authorization Acknowledgment/Status (845)	
	PB: Railroad Parameter Trace Registration (455)	
	PC: Purchase Order Change Request - Buyer Initiated (860)	
	PD: Product Activity Data (852)	
	PE: Periodic Compensation (256)	
	PF: Annuity Activity (268)	
	PG: Insurance Plan Description (100)	
	PH: Pricing History (503)	
	PI: Patient Information (275)	
	PJ: Project Schedule Reporting (806)	
	PK: Project Cost Reporting (839) and Contractor Cost Data Reporting (196)	
	PL: Railroad Problem Log Inquiry or Advice (452)	
	PN: Product Source Information (244)	
	PO: Purchase Order (850)	
	PQ: Property Damage Report (112)	
	PR: Purchase Order Acknowledgment (855)	
	PS: Planning Schedule with Release Capability (830)	
	PT: Product Transfer and Resale Report (867)	
	PU: Motor Carrier Shipment Pickup Notification (216)	
	PV: Purchase Order Shipment Management Document (250)	
	PW: Healthcare Provider Information (274)	
	PY: Payment Cancellation Request (829)	
Q	QG: Product Information (878, 879, 888, 889, 896)	
	QM: Transportation Carrier Shipment Status Message (214)	
	QO: Ocean Shipment Status Information (313, 315)	

Group	Functional ID Code (Table)	Notes
R	RA: Payment Order/Remittance Advice (820) RB: Railroad Clearance (470) RC: Receiving Advice/Acceptance Certificate (861) RD: Royalty Regulatory Report (185) RE: Warehouse Stock Receipt Advice (944) RF: Request for Routing Instructions (753) RG: Routing Instructions (754) RH: Railroad Reciprocal Switch File (433) RI: Routing and Carrier Instruction (853) RJ: Railroad Mark Register Update Activity (434) RK: Standard Transportation Commodity Code Master (435) RL: Rail Industrial Switch List (423) RM: Railroad Station Master File (431) RN: Requisition Transaction (511) RO: Ocean Booking Information (300, 301, 303) RP: Commission Sales Report (818) RQ: Request for Quotation (840) and Procurement Notices (836) RR: Response to Request For Quotation (843) RS: Order Status Information (869, 870) RT: Report of Test Results (863) RU: Railroad Retirement Activity (429) RV: Railroad Junctions and Interchanges Activity (437) RW: Rail Revenue Waybill (426) RX: Rail Description (432) RY: Request for Student Educational Record (Transcript) (146) RZ: Response to Request for Student Educational Record (Transcript) (147)	RN: Transaction set for ordering equipment and material. Can also be used to inquire about, change, or terminate the original order.

Group	Functional ID Code (Table)	Notes
S	SA: Air Shipment Information (104)	
	SB: Rail Carrier Services Settlement (424)	
	SC: Price/Sales Catalog (832)	
	SD: Student Loan Pre-Claims and Claims (191)	
	SE: Shipper's Export Declaration (601)	
	SH: Ship Notice/Manifest (856)	
	SI: Shipment Information (858)	
	SJ: Transportation Automatic Equipment Identification (160)	
	SL: Student Aid Origination Record (135, 139)	
	SM: Motor Carrier Load Tender (204)	
	SN: Rail Route File Maintenance (475)	
	SO: Ocean Shipment Information (304, 309, 311, 317, 319, 322, 323, 324, 325, 326, 350, 352, 353, 354, 355, 356, 357, 358, 361)	
	SP: Specifications/Technical Information (841)	
	SQ: Production Sequence (866)	
	SR: Rail Carrier Shipment Information (404, 419)	
	SS: Shipping Schedule (862)	
	ST: Railroad Service Commitment Advice (453)	
	SU: Account Assignment/Inquiry and Service/Status (248)	
	SV: Student Enrollment Verification (190)	
	SW: Warehouse Shipping Advice (945)	

Group	Functional ID Code (Table)	Notes
T	TA: Electronic Filing of Tax Return Data Acknowledgment (151) TB: Trailer or Container Repair Billing (412) TD: Trading Partner Profile (838) TE: Tax or Fee Exemption Certification (283) TF: Electronic Filing of Tax Return Data (813) TI: Tax Information Exchange (826) TJ: Tax Jurisdiction Sourcing (158) TM: Motor Carrier Delivery Trailer Manifest (212) TN: Tax Rate Notification (150) TO: Real Estate Title Services (197, 199, 265, 485, 486) TP: Rail Rate Transactions (460, 463, 466, 468, 485, 486, 490, 492, 494) TR: Train Sheet (161) TS: Transportation Services Tender (602) TT: Educational Testing and Prospect Request and Report (138) TU: Trailer Usage Report (227) TX: Text Message (864)	
U	UA: Retail Account Characteristics (885) UB: Customer Call Reporting (886) UC: Secured Interest Filing (154) UD: Deduction Research Report (891) UI: Underwriting Information Services (255) UP: Motor Carrier Pickup Manifest (215) UW: Insurance Underwriting Requirements Reporting (186)	
V	VA: Vehicle Application Advice (126) VB: Vehicle Baying Order (127) VC: Vehicle Shipping Order (120) VD: Vehicle Damage (124) VE: Vessel Content Details (109) VH: Vehicle Carrier Rate Update (129) VI: Voter Registration Information (280) VS: Vehicle Service (121)	

Group	Functional ID Code (Table)	Notes
W	WA: Product Service Transaction Sets (140, 141, 142, 143)	
	WB: Rail Carrier Waybill Interchange (417)	
	WG: Vendor Performance Review (501)	
	WI: Wage Determination (288)	
	WL: Well Information (625)	
	WR: Shipment Weights (440)	
	WT: Rail Waybill Request (425)	

Version Number

Version Number is the code specifying the version number of the interchange control segments.

Number	Description
00200	ASC X12 Standards Issued by ANSI in 1987
00201	Draft Standards for Trial Use Approved by ASC X12 Through August 1988
00204	Draft Standards for Trial Use Approved by ASC X12 Through May 1989
00300	ASC X12 Standards Issued by ANSI in 1992
00301	Draft Standards for Trial Use Approved for Publication by ASC X12 Procedures Review Board Through October 1990
00302	Draft Standards for Trial Use Approved for Publication by ASC X12 Procedures Review Board Through October 1991
00303	Draft Standards for Trial Use Approved for Publication by ASC X12 Procedures Review Board Through October 1992
00304	Draft Standards for Trial Use Approved for Publication by ASC X12 Procedures Review Board through October 1993
00305	Draft Standards for Trial Use Approved for Publication by ASC X12 Procedures Review Board through October 1994
00306	Draft Standards for Trial Use Approved for Publication by ASC X12 Procedures Review Board through October 1995

Number	Description
00307	Draft Standards for Trial Use Approved for Publication by ASC X12 Procedures Review Board through October 1996
00400	ASC X12 Standards Issued by ANSI in 1997
00401	Standards Approved for Publication by ASC X12 Procedures Review Board through October 1997
00402	Standards Approved for Publication by ASC X12 Procedures Review Board through October 1998
00403	Standards Approved for Publication by ASC X12 Procedures Review Board through October 1999
00404	Standards Approved for Publication by ASC X12 Procedures Review Board through October 2000
00405	Standards Approved for Publication by ASC X12 Procedures Review Board through October 2001
00406	Standards Approved for Publication by ASC X12 Procedures Review Board through October 2002
00500	ASC X12 Standards Issued by ANSI in 2003
00501	Standards Approved for Publication by ASC X12 Procedures Review Board through October 2003

OBMSGID

The `OBMSGID` key value pair is available in both `Reply_Gen` and `IB_Reply` TPS when **Await Replies** is selected. `OBMSGID` can be explicitly appended to `dispList`. If not, then the reserved OB message is removed from both the memory and Recovery database when the IB reply comes in and the await reply mode of the protocol is shut down.

When **Awaits Reply** is chosen and UPoCs are configured in the inbound reply TPS, the `OBMSGID` key is introduced into TPS in RUN mode. The `OBMSGID` key holds reserved OB messages.

In the IB reply TPS the string is:

```
ibreplyUPoC {MSGID message0} {OBMSGID message1} {CONTEXT sms_ib_reply} {ARGS {}} {MODE run} ...
```

A typical RUN mode of the `ib_reply` UPoC could be similar to:

```
roc ibreplyUPoC {args} {
  keylget args MODE mode      ;#Fetch mode
  set dispList {}             ;#Nothing to return
  switch -exact -- $mode {
    :
  }
  run {
```

```

        # 'run' mode always has a MSGID; fetch and process it
        keylget args MSGID mh
        keylget args OBMSGID ob_mh
        lappend dispList "CONTINUE $mh"
        if { [info exist ob_mh]} {
            lappend dispList "KILL $ob_mh"
        }
    }
    :
}
return $dispList
}

```

Another example of how this argument can be used is:

```

proc hcitpsobresend4ibreply {args} {
    keylget args MODE mode ;# Fetch mode
    set dispList {} ;# Nothing to return
    switch -exact -- $mode{
        start {
            #Perform special init functions
            #NB.: there may or may not be a MSGID key in args
        }
        run {
            # 'run' mode always has a MSGID; fetch and process it
            keylget args MSGID
            keylget args CONTEXT ctx
            keylget args OBMSGID ob_mh
            if {[info exist ob_mh]} {
                if { $ctx == "reply_gen" } {
                    if {[msglength $ob_mh] != 0} {
                        #Resend reserved OB message when IB Reply comes late
                        lappend dispList "PROTO $ob_mh";
                        lappend dispList "KILL $mh";
                    } else {
                        # 0-length control message should not be resent, shutdown
                        lappend dispList "KILL $ob_mh";
                        lappend dispList "CONTINUE $mh";
                    }
                } elseif { $ctx == "sms_ib_reply" } {
                    #IB Reply comes in
                    lappend dispList "KILLREPLY $mh";
                    lappend dispList "KILL $ob_mh";
                } else {
                    # When there is no reserved OB message.
                    lappend dispList "CONTINUE $mh"
                }
            }
            time {
                #Timer-based processing
                # N.B.: there may or may not be a MSGID key in args
            }
            shutdown {
                #Doing some clean-up work
            }
        }
    }
    return $dispList
}

```

Properties tab

The **Thread**, **Inbound**, and **Outbound** tabs contain the configurations for the protocol thread.

The **Properties** tab (default) distinguishes each thread (connection) from other threads and to which process each thread belongs.

This table shows property parameters:

Parameter	Description
Thread Name	<p>This is the connection name that displays on the thread icon.</p> <p>The thread name is also used as the embedded recovery/error database's user name. Because of this, there is a possible conflict when a configured thread has the same user name used by the <code>hcidbdump</code> utility. If you must have a thread named <code>_hcidbdump_</code>, then run <code>hcidbdump</code> with the <code>-U</code> option to select an alternate user name to use for <code>hcidbdump</code>.</p>
Bitmap	<p>This is the file name that contains the bitmap for the connection icon. This is the image that displays on the thread icon. Click List to specify a bitmap in another file.</p>
Process Name	<p>This is the name of the engine process that controls this thread. The menu lists all configured processes.</p>
Group Names	<p>This is for the names of the thread groups to which this thread belongs. Use thread groups to control and monitor related threads. One or more strings can be used to limit views.</p>
Engine Log Configuration	<p>This is the alias for Engine Output (EO) configuration information. This controls how much output to write to the process log for this thread. Click List to open the EO Configuration selection dialog box.</p>
Protocol	<p>This is the communication protocol for the thread. Click the arrow to open a list of available formats. Then, click Properties to configure the protocol properties for that format.</p> <p>Icons shown are based on the selected thread protocol. After a protocol is changed in the thread properties, the icon is also changed. If you have already specified a custom icon in the thread properties, then the custom icon is used regardless of the selected protocol.</p>

Parameter	Description
Encoding	<p>The selection in this field applies to inbound and outbound messages. Inbound messages are converted from the specified encoding to Unicode and outbound messages are converted from Unicode to the specified encoding.</p> <p>You can configure the menu list by editing the [encoding] list in %HCIR00T\server\i18n.ini.</p>
XML Encoding	<p>This applies to inbound and outbound messages. It indicates that the encoding of each message is dynamically determined from an XML declaration in the message. If XML Encoding is selected and no XML declaration is found in a message, then UTF-8 encoding is assumed. This is the default for XML documents.</p>
HL7 Encoding	<p>This applies to inbound and outbound messages. It indicates that the encoding of each message is dynamically determined from value of MSH.18 in the HL7 message. If HL7 Encoding is selected and no value of "MSH.18" is found in a message, then UTF-8 encoding is assumed. This is the default for HL7 message.</p>
Inter-Site Routing Port	<p>Inter-site routing routes to a destination thread that is located in another site and even on another host. See Inter-site routing.</p>
Error Database Procs	<p>The error database TPS gives you a control point before a message is written to the error database. This means you have the opportunity to change the error message content and to route the message to another context.</p>
Retries	<p>This is the maximum number of times to attempt message transmission:</p> <ul style="list-style-type: none">• "0" is the initial transmission only.• "1" is the initial transmission plus one retry.• "-1" is unlimited retries.
Auto-Start Connection	<p>Select this to start the thread automatically whenever the specified engine process is started.</p>

Parameter	Description
Use Recovery Database	<p>The use of the recovery database is determined at the inbound level. This controls saving inbound or outbound messages, and inbound message logging. The engine does not use the recovery database when the outbound thread is checked and the inbound thread is unchecked. Any file name can be entered. After a message is marked for recovery, it is backed by the recovery database throughout its life in the engine.</p> <p>Messages that are tagged for disk-based queuing are backed by the recovery database throughout their life in the engine.</p> <p>The recovery database backs up the thread and process queues. If an engine process or thread stops running, then it retains the message data in the recovery database when the thread or process is restarted.</p>
Enable message tracing (thread level)	<p>When this is selected, tracing information is saved with the message trace framework to the tracing database.</p> <p>Message tracing is enabled on the process level or thread level.</p> <p>If process level tracing is enabled, then there is no requirement to turn on tracing for each individual thread under the process. If process level tracing is disabled, then tracing for a thread is enabled with the thread level option.</p> <p>In thread level tracing, selecting this turns on tracing for a specific thread.</p>
Messages routed to this connection treated as inbound	<p>Select this to make the destination behave as inbound, not outbound, so that you can route messages that are delivered to the destination site.</p>

Inbound pane

This table shows the configurable inbound pane parameters:

Parameter	Description
Use byte order mark	<p>This applies to inbound messages only. A byte order mark is a pre-defined sequence of bytes at the start of a message that defines both the byte order and the encoding. When a byte order mark is found, it overrides any encoding specified in the Encoding field. A byte order mark also overrides an XML declaration in XML mode. The byte order mark is not required, and if one is not found, Encoding and XML Encoding operate as they should.</p> <p>Supported byte order marks are:</p> <ul style="list-style-type: none">• FE FF Endian Order: Big Encoding: UTF-16BE• FF FE Endian Order: Little Encoding: UTF-16LE• EF BB BF Endian Order: N/A Encoding: UTF-8 <p>Big Endian, or BE, means that the most significant bytes happen first and the least significant bytes happen last. Little Endian (LE) is the reverse.</p> <p>The byte order does not apply to UTF-8 encoding, although the UTF-8 byte order mark is valid.</p> <p>Because byte order marks are valid for more esoteric encodings similar to UTF-7, these are extremely rare and are not supported.</p> <p>The byte order mark is stripped from the inbound message before encoding conversion.</p>

Parameter	Description
Save Inbound Messages	<p>Click this to save inbound message data to a SMAT database file. Messages are saved to this file before any further work in the engine. This provides a backup of the data as it was received from the sending system.</p> <p>You can configure the file name to populate with a default value by configuring the default value on the Client Preferences NetConfig/NetMonitor tab. Click Advanced to open the Configure Default Values dialog box. Select an option from the menu, or specify one in the field.</p> <p>For example, a thread named <code>conn_1</code> has values of <code>@threadname@_in</code> and <code>@threadname@_out</code>. When Save Inbound Messages is selected, the file names automatically populate with <code>conn_1_in</code> and <code>conn_1_out</code>.</p>
Save inbound resent messages	<p>Select this to save inbound resent messages to the SMAT Database file.</p> <p>This setting is related to the selection of Save inbound messages. This option is enabled only when Save inbound messages is selected.</p> <p>For this option, the <code>INSAVERESENTMSGTOSMAT</code> key is in the NetConfig file.</p> <p>For additional information, see the "Save inbound resent messages" after this table.</p>
File	<p>Specify the file name for the SMAT file, if Save Inbound Messages is selected. This file is located in <code>%HCISITEDIR\exec\processes\processname\smat file name</code>.</p>
Message Archiving	<p>Archives the inbound message after SMAT messages are saved. Both inbound and outbound messages are archived in UTF-8 encoding, but not the original encoding.</p> <p>These messages should not be used for resending to system threads.</p> <p>Message archiving is independent from SMAT.</p> <p>When Message Archiving is selected, Setup is enabled. Clicking this opens the Setup Message Archiving dialog box.</p>
Msg Engine Log Configuration	<p>This is the engine output alias that describes how much output to write to the process log for each message. Click List to open the In Message EO Config selection dialog box.</p>

Saving inbound resent messages

The **Save inbound resent messages** function has been added to all **Thread Properties** and **Resend** dialog boxes. The thread properties setting takes precedence.

Note: Users decide if messages are traceable, or not.

This option is available only when resending messages to the engine, and not a file.

Usage:

- In the **Thread Properties** dialog boxes, when you select **Save inbound resent messages**, resent messages are saved and the dialog box option is disabled. Users cannot change this setting.
- In the **Thread Properties** dialog boxes, when you do not select **Save inbound resent messages**, the dialog box option is enabled. Select this to save the resent message.

Save inbound resent messages only takes effect when resending to inbound pre-TPS.

Resending to other queues is out-of-scope based on the current implementation of this flag.

Outbound pane

Parameter	Description
Generate byte mark order	<p>This applies to outbound messages only. The encodings that are listed in the previous table have a byte order mark inserted into the outbound message when this is selected. This setting is ignored for other encodings.</p> <p>This has no effect on which outbound encoding is used. It indicates that additional bytes are added to the messages when the encoding is appropriate. The outbound encoding is defined by Encoding and XML Encoding.</p>
Save Outbound Messages	<p>Click this to save outbound message data to a SMAT database file. Messages are saved to this file before sending the message to the receiving system. This provides a backup of the data.</p> <p>You can configure the file name to populate with a default value by configuring the default value on the Client Preferences NetConfig/NetMonitor tab. Click Advanced to open the Configure Default Values dialog box. Select an option from the menu, or specify one in the field.</p> <p>For example, a thread named <code>conn_1</code> has values of <code>@threadname@_in</code> and <code>@threadname@_out</code>. When Save Outbound Messages is selected, the file names automatically populate with <code>conn_1_in</code> and <code>conn_1_out</code>.</p>

Parameter	Description
File	Specify the file name of the SMAT file, if Save Outbound Messages is selected. This file is located in %HCISITEDIR\exec\processes\processname\smat file name.
Message Archiving	<p>This archives the outbound message after SMAT messages are saved and after the pre-write TPS has finished. Outbound messages are archived in UTF-8 encoding, not the original encoding. For the Multi-Byte version, the messages are archived after the change from the UTF-8 internal format to outbound encoding.</p> <p>These messages should not be used for resending to system threads.</p> <p>Message archiving is independent from SMAT.</p>

Multi-byte order options

Use byte order mark and **Generate byte order mark** are always valid and may be selected in any combination. Some combinations of settings may seem nonsensical. For example, selecting **Generate byte order mark** with Big5 encoding does not have any effect. These are permitted as they do no harm and are difficult to prevent.

If any unicode encoding or XML is selected, then it looks for a BOM. If the user selection does not contain UTF-8 or XML is unchecked, then it does not look for the BOM.

If the BOM is found, then it overrides XML and the user selection. If no BOM is found and XML is selected, then it looks for the XML encoding. If XML is not checked, then it defaults to the user selection. If no user selection exists, then it is UTF-8.

The order is **BOM > XML > user**.

Inbound tab

This table shows the **Inbound** tab's parameters:

Parameter	Description
Outbound Only	<p>Click this to make the connection send-only. The engine receives only reply messages from the host. Clicking this disables all other configuration entries in this pane.</p> <p>When this is selected, the engine expects to receive a reply if you have set up the thread to do so. By setting up for replies, the outbound thread sets an awaiting reply flag each time it sends a message out. This happens when Await Replies is selected on the Outbound tab. This tells the engine to expect a reply. Outbound Only ignores any unexpected inbound messages that arrive when the awaiting reply flag is not set.</p> <p>When this is cleared, this indicates that you are using your outbound thread to receive data messages from the destination system. An example would be a request for a lab order number. A system could be set up to respond back to the outbound thread with this lab order number. This is not common because most interfaces use another inbound thread to communicate data back to the source system.</p> <p>If you are having problems with losing replies from the destination system, then you can temporarily clear this option. In this way, you can track replies that may have been received after the reply time-out period. After the time-out is met, the awaiting reply flag is turned off and this option discards late arriving replies.</p>
Hold	<p>Select this to hold inbound data. When this is cleared, inbound data is released.</p> <p>When selected, messages can be resent to a running inbound thread. Simultaneously, this option holds the thread from reading messages through its protocol driver. This is required to maintain the correct message order when resending</p>

Parameter	Description
TPS Inbound Data	<p>Click Edit to open the Inbound TPS Editor. Then, click Add to open the TPS Properties dialog box, from where you can select the procedures or Java class and arguments for inbound data messages. This TPS pre-processes inbound data messages before being sent to a translation thread.</p> <p>If a procedure applies to all messages, then put the procedure in TPS Inbound Data. If the procedure applies to less than all messages, then put the procedure in one or more route details.</p> <p>When Run in a Sub-thread is selected on the Inbound TPS Editor, a sub-thread is created under the protocol thread. This sub-thread is used only for data type messages, not those of type reply. There must be a TPS defined; there is no sub-thread if a TPS is not defined.</p>

Parameter	Description
TRx ID Determination Format	<p>Click the arrow to select the format from the menu. The format is used to parse the record into fields for later translation. Then, click Edit to open the Trx ID determination dialog box. Use this dialog box to configure transaction ID information:</p> <ul style="list-style-type: none"> • Selecting DICOM, disables Edit. There are no properties to edit. The trxid for the DICOM protocol is <i>AbstractSyntax_CommandCode</i>. For example: 1.2.840.10008.5.1.4.31_0020, Patient Root Query/Retrieve Information Model_MOVE_C_MOVE_RQ. • For FRL, select Offset and Length. These specify the offset and length to the location of the Trx ID in the record. • For HL7, HPRIM, X12, NCPDP, and UN/EDIFACT, it is not necessary to select the version and variant. Edit is not available when any of these are selected. • For JSON, it selects the first type field value to identify the Trx ID of the message/transaction. Use field routing to select a key or rely on the default first value that is encountered in the json structure as the Trx ID. • Selecting XML disables Edit. There are no properties to edit. • Selecting XML (Namespace) disables Edit. There are no properties to edit. • Select UPOC to specify a UPoC without going through FRL. Click Edit to open the Format UPoC dialog box. Only one proc can be selected. Specify arguments for inbound data messages. TPS configurations are supported when the format is UPoC.

Parameter	Description
EDI Batch	<p>Click the arrow to select the format from the list. Then, click Edit to open the batch configuration dialog box.</p> <p>This feature requires an additional license to be functional.</p> <ul style="list-style-type: none"> For NCPDP Telecom, select the version. If EDI Batch type NCPDP Telecom is specified for an inbound thread, then all inbound messages are treated as batches. For X12, select the version and whether to split to Groups or Transaction Sets. As the inbound NCPDP Formulary and Benefit batch has no parameters, Edit is unavailable.

Run in a sub-thread option

When this option is selected on the Inbound TPS Editor by clicking **Edit** for **TPS Inbound Data**, a sub-thread is created under the protocol thread. This sub-thread is used only for data type messages, not those of type reply. There must be a TPS defined; there is no sub-thread if a TPS is not defined.

There is only one thread involved with any IB/OB thread SMS (that is, TPS).

All protocol threads have a pre-SMS queue and a post-SMS queue. If there is a TPS defined, then messages go in the pre-SMS queue. The pre-SMS queue is skipped if no TPS is defined for the protocol thread. Between the two queues is the TPS that does something with the message and generates more messages.

If a sub-thread is used, then that TPS runs in it. During the time of the TPS processing, the engine is free to do other work. For example, input messages, output messages, move messages as part of the data flow. When the sub-thread has run the TPS procedure, it then signals the engine to pass the results into the post-SMS queue.

If the defined TPS is small and fast, then there is no requirement to use a sub-thread. The overhead of starting the thread would negate any gain. By “fast,” it is meant that it does not parse a message, as that can take time. It also does not access external data.

In terms of stages, the IB pre and post queues are stages 1 and 2 and the OB queues are stages 10 and 11.

In general, a single CPU machine gains slightly or even loses by turning on the sub-thread. This is the same as xlates.

In the same way, a process that is lightly loaded does not gain anything using the sub-thread. There is no work for the engine to do when it runs.

A sub-thread is helpful when there is a CPU available. It is also helpful when there is other work for the engine to do when the sub-thread is running. Parallel processing of messages is permitted if that is possible in the configuration.

Note: When using Java TPS UPoCs, the **Inbound** tab shows the procedure as cljTPS. This is the Tcl interface procedure to Java TPS. When you click **Edit**, the Java class is shown. Java UPoCs cannot be used if **Run in a Sub-thread** is enabled.

Using globals

If you have TCL procedures that set globals in the global namespace, then it helps to know how your message flow is affected when using globals.

For example, the first message arrives in the TCL (TPS) for the Inbound TPS UPoC. A global is set to check the next message for status. The second message arrives in this same TCL procedure and the logic examines if this is the expected next message.

There is only one SMS interpreter that does the TPS per protocol thread. This is true even if threaded SMS is not enabled.

When threaded, the difference is that the work in this interpreter is performed in a sub-thread. In this case, globals work fine and anything that is performed in the startup mode stays around. This is per protocol thread, so the IB and OB use other ones.

This differs from threaded xlates where they can bounce around depending on the options selected.

The threaded option only runs if the message type is data. The same interpreter is used when you have a reply, but it is not threaded.

Outbound Replies pane

This table shows the configurable parameters for outbound replies:

Parameter	Description
Hold	This holds the transmission of all outbound reply messages until released using Network Monitor.
TPS Outbound Reply	Click Edit to open the TPS Editor . Then, click Add to open the TPS Properties dialog box, where you can select the procedures or Java class and arguments for outbound reply messages.
Retries	<p>This is the maximum number of times to attempt transmission of an inbound message to the destination thread. A retry happens when a protocol write fails.</p> <ul style="list-style-type: none"> 0 is the initial transmission only. 1 is the initial transmission plus one retry. -1 is unlimited retries.
Interval	This is the number of seconds to wait between each retry.

Parameter	Description
Send OK Procs	Click Edit to open the TPS Editor . Then, click Add to open the TPS Properties dialog box. In the dialog box, select the procedures or Java class and arguments to run upon successful transmission of the inbound message to the destination thread.
Send Fail Procs	Click Edit to open the TPS Editor . Then, click Add to open the TPS Properties dialog box. In the dialog box, select the procedures or Java class and arguments to run upon unsuccessful transmission of the inbound message to the destination thread. An unsuccessful transmission happens when the number of retries fails. All retries and forwarding threads are exhausted before a transmission is treated as failed.
Prewrite Procs	Click Edit to open the TPS Editor . Then, click Add to open the TPS Properties dialog box, from where you can select the procedures or Java class and arguments. These procedures provide user-control immediately before the protocol write happens. The only dispositions that are permitted in the prewrite are "CONTINUE" and "KILL".
Forward Thread	Click List to open the Threads list box. Use the list box to select the thread to serve as an alternate destination for the outbound replies. The engine first exhausts the permissible number of transmission retries to the primary destination thread before forwarding the message to the alternate thread.

Field routing

When you select **Field Routing** on the **Trx ID Determination Format** menu on the **Inbound** or **Outbound** tab, clicking **Edit** opens the **Trx ID Determination** dialog box.

Click **Select** to open the **Select Record Format** dialog box.

OK, the dialog box is populated with the format tree. The format tree is where you can select field addressing.

Based on the selection in the tree, the field address is added to the bottom field when you click **Set Field Address**. You can also double-click the field to add it to **Field Address**. When you make a selection and click **.** The field address is editable. You can also specify the field address directly into the text box.

Saving field routing settings

The field routing for Trx ID Determination is saved in the DATAFORMAT section of a thread protocol section:

Single address:

```
{DATAFORMAT {
  { GRMADDR 0(0).MSH(0).#9(0) }
  { MSG {hl7 2.2 22 DFT} }
  { TYPE grm }
} }
```

Multiple addresses:

```
{DATAFORMAT {
  { GRMADDR {0(0).MSH(0).#9(0) 0(0).MSH(0).#6(0)} }
  { MSG {hl7 2.2 22 DFT} }
  { TYPE grm }
} }
```

- TYPE is grm.
- GRMADDR is the selected field address(es) which specifies the message.
- MSG is not currently used by the engine. It is used to save the selected Format, Version, Variant, and Message. This is used by the IDE to show the last selection on Field Routing Trx ID Determination.

If the variant name or other specification that was saved in NetConfig no longer exists, then no variant or default specification is selected. This applies when the format is selected the first time.

Multiple addresses

Multiple addresses are also supported. To select multiple addresses, use **Ctrl-click**.

Click **Set Field Address** to place the selected addresses into the **Field Address** field.

Selecting child nodes under the same parent node are presented as one field address.

The selected field address(es) settings are saved in the NetConfig file. The **Field Routing** dialog box for Trx ID Determination shows the loaded settings, except for the selection status on the tree.

For multiple addresses, when an error condition exists, the engine attempts to process the other GRM addresses.

For example:

```
{ GRMADDR {0(0).MSH(0).#20(0) 0(0).MSH(0).#6(0)} }
{ MSG {hl7 2.2 22 DFT} }
{ TYPE grm }
```

The GRM address 0(0).MSH(0).#20(0) is invalid and 0(0).MSH(0).#6(0) is valid.

In this case, the engine does the xlate for the TrxID for 0(0).MSH(0).#6(0), such as {HCI}.

Route Replies tab

The **Route Replies** tab shows all reply routes which start from the selected thread in the tree table.

If one thread is selected, then this tab shows all reply routes which start from the selected thread.

If one route is selected, then the **Transaction Summary** tab opens.

Double-click a row in the table to open the **Route Details** dialog box.

Right-click a row to open a menu of options.

Route Messages tab

The **Route Messages** tab shows routes for the currently selected thread or route in a tree table.

Routes define how a message proceeds through the engine. Configure routes after creating and arranging the connection icons on the main dialog box.

If one thread is selected on the Network Graph panel, then this tab shows all routes which start from the selected thread.

If one route is selected, then the **Transaction Summary** tab opens.

Route details

Editing buttons let you conveniently add routes, details, cut, copy, and so on.

Right-click a row to open a context menu of editing options.

Double-click a row in the table to open the **Route Details** dialog box.

Notes tab

The thread **Notes** tab shows user-added notes.

In the Network Configurator, thread notes are added through the **Notes** tab. This tab contains a text area that is used to add notes for a thread.

A visual indicator on the thread icon tells you that there is a note for that thread.

Process notes are added through the Network Configurator's **Process Configuration** dialog box.

Note: The Notes file lock respects the lock for the NetConfig file. For the revision function in the NetConfig file, the Notes file does not have revision. So, there are no notes for the revision NetConfig file. Because of this, if the revision NetConfig file is selected, there is no Notes tab on the **Thread Properties** dialog box or the **Process Configuration** dialog box.

Viewing notes in Network Monitor

Notes can be viewed in Network Monitor. The Network Monitor's Notes dialog box shows the notes for the selected process or thread.

To view the thread or process notes in the Network Monitor, select the appropriate **Notes** menu item. This is accessed by right-clicking the thread or process.

If there are no notes for a thread or process, then the **Notes** option is unavailable.

When you select **Notes**, a read-only dialog box opens to show the notes which were added and saved in Network Configurator.

Notes HTML features

The HTML that is viewed and is used in the IDE supports HTML 3.2. This only renders basic HTML. It cannot embed applets or Flash. It cannot interpret JavaScript.

Notes file

When notes are added to a process or thread, a new folder is created in the site directory.

For every process that has process notes or has at least one thread notes, a *process_name.notes* file is created for the process notes.

Similarly, the thread notes file is *thread_name.notes*.

Notes are located under `$HCIR00T\SiteDir\notes\netconfig_file_name\process_name\notes_file`.

The thread notes file, process notes file, and process directory are created as required and removed if there are no related notes.

All process notes and thread notes files are plain text and you can edit them by any text editor. The notes file is always saved and read based on UTF-8 encoding.

If you use an external editor to edit the notes file and save it as non-UTF-8 encoding, then the IDE cannot read it properly. This does not affect notes with ACSII- only content because UTF-8 covers ASCII characters.

Verifications tab

If a property is incorrect, then when **Apply** is clicked an error message displays and the **Verifications** tab lists the errors.

If **Verify Thread** is clicked first and an error exists, then the dialog box automatically goes to the **Verifications** tab with a list of all errors.

Transaction Summary tab

The **Transaction Summary** tab displays when you click a route line, showing a summary of the route information.

Creating routes

Routes are created according to the type of thread.

- 1 In Network Configurator, select a protocol and drag it onto the layout panel.
- 2 Right-click the new thread and select **Create Route** or **Create Reply Route**.
- 3 Move the mouse to the destination thread and click. This creates a route line between the two threads, and a **Create Message** dialog box opens for configuring route details.

Before the target thread is selected, during the dragging you can interrupt the route creation by pressing the **ESC** key. You can also click in an open space.

- 4 In the **Name** field, specify a transaction ID name.

The existing route names from the same source thread are listed on the menu. If there is no existing route name, then you can select **New Id** or use a new name.

The transaction ID indicates the type of message. This is the type of transaction that is represented by the record. The route name must match the Trx ID in the record, including case.

For message-type record formats, specify the name of the message type.

For fixed-length record layouts, specify the exact characters that display in the transaction ID field of the inbound data message.

- 5 If you do not select a name, then you must select **Static Route** or **Static else route**. If you select an existing route name, then the current route belongs to the selected route name.

A static route processes all messages the same way, regardless of their transaction ID. Use a static route to apply a common action to all transactions. For example, sending a copy of all messages to an archival site.

A Static Else Route is similar to a Static Route. This option creates the `_HCl_static_else_route_`.

- 6 Select additional parameters, as required.

- **Wild Card Route** routes a set of messages to one destination. For example, all ADT messages to one destination would use a wildcard: `ADT.*`.

This setting can be shared by multiple routes. If the shared setting is changed, then a warning message opens after you click **OK**: The changed Wild Card Route is shared by the routes: `raw1`, `raw2`. This change will be applied to all of them.

Name behaves as a regular expression. For a message to be considered a match, its transaction ID has to be a complete match to this expression.

For example, the wildcard `ADT_A0\d` is the equivalent of the Tcl regexp `{^ADT_A0\d$}` `$transactionID`. This wildcard matches a message whose transaction ID is `ADT_A01`, but does not match a message whose transaction ID is `ADT_A17` or `ADT_A01a`.

If negative lookahead is used in the wildcard route, then the expression should be written properly. For example, to match a route started with `ADT_A` and it is not ended in 17 with negative lookahead expression, `ADT_A(?:!17)\d\d` should be used. `ADT_A(?:!17)` does not work.

To use regular expression shorthand in wildcard route, use double slashes. For example, to make the above negative lookahead expression work as wildcard route, the correct **Name** value is `ADT_A(?:!17)\\d\\d`.

- Select **Remove all messages when any route detail fails** to define route or reply actions when an error happens in a route detail. When this is selected, all messages are dropped. Otherwise, only the error route detail is dropped.

- Selecting **Store original message in metadata for downstream processing** caches the pre-xlate message in Route Details and is used in the **Database-Outbound Protocol Properties** dialog box and Database-Outbound protocol engine thread.

When this is selected, the engine thread caches the pre-xlate message at runtime. By default, this is cleared.

- **Enabled** enables/disables the route.

7 Configure the Detail Properties parameters. For these properties, see the individual types:

- [Configuring a raw route detail.](#)
- [Configuring a generate route detail.](#)
- [Configuring an xlate route detail.](#)
- [Configuring an XSLT route detail.](#)
- [Configuring a Chain route detail.](#)
- [Configuring a Branch route detail.](#)

Route configuration

Route configuration gives you the ability to improve usability for message routes and related configuration.

To enter this mode, select a thread on the Network Graph panel. Then, click **Configure Thread Routes**. This contains a toolbar, message route graph panel, and properties panel.

Configure Thread Routes toolbar

The toolbar contains buttons for switching back to the Network Configuration panel and buttons for adding nodes to the Message Route Graph panel.

- **Configure Netconfig** switches back to the Network Configuration panel.
- **Configure Message Routes** switches between the Configure Thread Routes panel for the current thread.
- **Configure Reply Routes** switches between the Reply Route Configuration panel for the current thread.
- **Trx ID** adds a transaction ID to the current message flow. To do this, drag and drop this button over a thread node on the Message Route Graph panel, which adds a transaction ID node.

Use these buttons for a route type node after a transaction ID node. To add a route type node, drag and drop a button over a transaction ID node.

- **Raw** tells the engine to pass the message data through without translation. Alternately, use one or more Tcl procedures to process the message. In a raw route, other than pre-processors, the message is not interpreted or translated.
- **Xlate** is the typical translation mechanism. This translates the message according to the actions in a previously configured translation file. The configuration specifies the name of a translation specification (XLT) file to control the translation process.
- **Generate** is similar to raw, except that there is only the **Procs** option. This controls a message route using a single user-written Tcl procedure.
- **XSLT** takes XML messages as inputs and translates them into other formats according to the formats specified in the XSLT files. This is similar to Xlate, except that instead of specifying an Xlate file, you specify

a XSLT file. There is also no **Elevate warnings to errors** option on the **XSLT Route Details** dialog box. This is because XSLT is only used for XML and there is no elevate warning feature for XML.

- **Chain** permits you to mix Raw/Xlate/XSLT route types on one route detail.
- **Branch** does one common translation that is a Chain without a destination. Then, it branches to multiple translations which can be Raw, Xlate, XSLT, and Chain. This covers a common problem where a message must be normalized and then passed to several translations by TCP threads.
- **Route Configure Title** switches the title bar for each column (Protocol, Inbound Data, and so on).

Note: **Raw**, **Xlate**, **XSLT**, and **Chain** can also be dropped over a Branch route type node to create a branch.

Message route graph panel

This panel shows the starting point of route configuration.

The message flow starts with a single circle node in Protocol.

The Inbound Data node specifies the protocol between the inbound thread and the external system.

The thread node is the inbound node.

Note: The above nodes cannot be deleted by the user.

An indicator displays on nodes (inbound, outbound, and xlate) that have UPoCs associated with them (Java or Tcl).

The far right side of the panel lists all available threads which are used for adding destination threads in the message routes.

Reroutes

In the engine, a message is taken from the pre-xlate queue and routed to translation details using a TrxID. The details translate the message and send it on to the destination protocol thread.

Chains and branched chains only modify how and what the details do after routing; they do not permit a dynamic route after their translations.

Reroute adds this ability to specify another TrxID route after some translations of a message have taken place.

Routing happens when a message is taken from the pre-xlate queue. A reroute specifies that the output of a translation is placed in the front of the pre-xlate queue.

The action is the same as if a new message had come from the source thread and "went to the head of the processing line."

Optionally, you can also state a TrxID to be used when the rerouted message undergoes routing.

No rerouted message are routed to the static route TrxID.

Unlike single details, chains, and branched chains, reroutes are user-controlled using the post-xlate TPS procedure. There are no controls in the IDE for setting up the reroute.

Set the reroute in a TPS procedure or in the translation. Usually, if TPS is used, then it is expected that you use the post-xlate TPS to avoid errors.

Each message is checked to see if the user has requested a reroute after the translation of a detail has finished. If selected, then the message or a copy of the message is placed in the pre-xlate queue. Then, it undergoes routing in the normal manner.

If the recovery database is specified, then the message is placed in the database as if it were a new message from the source thread.

Generated routes can never use chains or branched chains but they may use the reroute feature.

Previously, a translation error in any detail would cause all translated messages under the active TrxID to be deleted.

This action is unchanged by the addition of chains, branched chains, or message reroutes. This also applies to reroutes that happen between the links of chains. They are placed in the partial queue and rerouted after all requested translations have been finished.

If the recovery database is enabled and there is an engine panic, then rerouted messages can be found in the database. This database is marked for the partial queue of the xlate thread or sub-thread.

These messages might have their reroute key as `REROUTED`, depending on how far processing has progressed at the time of the panic.

On the recovery during engine restart, all partially rerouted messages are processed as if the panic had not happened.

Rerouted messages that are in the database for the pre-xlate queue (state 5) have the reroute key `REROUTED`. They are recovered as if they were new messages from the source thread.

Reroutes happen at the end of each translation detail, after the post xlate TPS.

This permits a chain to produce several messages and a message to the destination protocol thread.

For example, a chain is "A" to "B" to "C" to "destination." Rerouting between "A" and "B" and between "B" and "C" results in two messages that are placed at the head of the pre-xlate queue (stage 5). The output of the "C" translation goes to the destination.

Both forms of setting reroute on a message require a key value. This value is used as the TrxID for routing of the message as it leaves the pre-xlate queue.

- This value can be a single TrxID or a Tcl list of TrxIDs.
- If the message is to undergo the normal TrxID determination set up for the source thread, then use the special value `NULL_REROUTE_TRXID` as the key value.
- This TrxID should never be specified for normal routes for the source thread.
- When this value is not used, the message is routed using the key's value. It does not use the normal TrxID determination processing.
- Using the key value for the TrxID improves performance by removing the overhead of its determination during routing.

A rerouted message is not processed by any static route specified for the message's source thread. To have a rerouted message routed by the static routes, use the normal static route TrxID, `_HCI_static_route_` as the reroute key's value.

When the number of xlate sub-threads for the process is nonzero, rerouted messages are processed according to the threaded xlate setup.

If the recovery database is in use and a recovery happens, then all actions are the same as normal processing.

That is, the reroute keys are saved as part of the metadata in the recovery database.

If a TPS procedure is used to set reroute on a message, then use the disposition of the same form as the other available dispositions. For example, `KILL` or `CONTINUE`. Similar to the existing dispositions, the reroute takes a message list, and each message must have a transaction ID.

Disposition form

The general form of the existing dispositions for more than one message is:

```
lappend dispList "CONTINUE {{$mh1 $mh2}}"
```

Brackets are used when there is more than one message. The reroute follows this general form but requires a more complex list structure as follows.

To reroute one message with a given TrxID:

```
lappend dispList "REROUTE {{$mh trxid_A}}"
```

To reroute two messages each with their own TrxID:

```
lappend dispList "REROUTE {{$mh1 trxid_A} {{$mh2 trxid_B}}"
```

Double brackets are used at the start of the message list.

As indicated, these are used even when the disposition is being set on a single message.

The TrxID cannot be defaulted.

Messages that are given the reroute disposition are not continued past the translation detail where it is set. To reroute a message and continue the message, you must make a copy of the message and set the correct dispositions.

Reroutes within the translation

Messages can also be rerouted during the translation using the `SEND` action. Because this is accomplished without adding a new action to the translations, it is controlled by a uniquely named local variable.

This example shows the form doing reroutes within the translation:

```
---- diddle fields ----
COMMENT Do reroute with trxid
=REROUTE trxid_A COPY @MSG_METADATA_REROUTE
SEND
---- diddle fields ----
COMMENT Do normal send operation
SEND
---- diddle fields ----
COMMENT Do reroute with two trxids
=REROUTE trxid_B trxid_C COPY @MSG_METADATA_REROUTE
SEND
--- diddle fields ----
COMMENT exit with a reroute using a trixid
=REROUTE trxid_D COPY @MSG_METADATA_REROUTE
--- diddle fields ----
COMMENT message exits translation in normal manner
```

The variable is reset when the `SEND` that is a reroute is completed.

In general, set the variable next to the `SEND` so that it is clear that it is not a normal `SEND`.

At least one TrxID should follow the word `REROUTE` in the string value of the variable `MSG_METADATA_REROUTE`.

As in the TPS form, if a message is to undergo normal routing then the TrxID should be set to the value `NULL_REROUTE_TRXID`.

Caution: Reroutes can form a closed loop. The engine cannot detect this condition.

Editing operations

Right-clicking a Type ID or route opens an editing menu.

This table shows the transaction editing operations:

Editing operation	Description
Disable/Enable Route	Disables/enables individual routes.
Add	Opens the Route dialog box for adding a new route at the end of the list.
Add Route After	Creates a new route and adds it after the current selection.
Add Route Before	Creates a new route and adds it before the current selection.
Add Route Detail	Opens the Route Details dialog box, where you can select a new Type and Properties.

This table shows the pertinent route editing operations:

Editing operation	Description
Add	Opens the Route Details dialog box, where you can select a new Type and Properties and places it after the current selection.
Add Route Detail After	Opens the Route Details dialog box, where you can select a new Type and Properties and places it after the current selection.
Add Route Detail Before	Opens the Route Details dialog box, where you can select a new Type and Properties and places it before the current selection.
Open Xlate File	Opens the selected <code>xlt</code> file in the Translation Configurator. This option displays when the route is xlate.

Protocol properties configuration

To configure this node, click **Configure Netconfig** to access the **Properties** tab.

See [Properties tab](#).

Thread inbound data configuration

This panel is shown when you select a thread node in the Route Graph panel.

See:

- [Inbound data pane](#)
- [Verifications tab](#)

Route detail property configuration

Click **Configure Thread Routes** to configure route detail properties for the selected route type node.

Add a Detail node by dragging a detail type from the toolbar and dropping it onto the transaction ID node.

An "X" on the icon represents an error. Click the **Verifications** tab for details.

On the **Details** tab, click **List** to open a list of available XSLT files. Click **Apply** to save the changes.

Add a destination node by dragging a thread from the thread list and dropping it over a route type node.

The **Details** tab content depends on the translation type:

- For a Raw node, see [Raw route detail](#).
- For an Xlate node, see [Xlate route detail](#).
- For a Generate node, see [Generate route detail](#).
- For an XSLT node, see [XSLT route detail](#).
- For a Chain node, see [Chain route detail](#).
- For a Branch node, see [Branch route detail](#).

Outbound thread properties configuration

Double-clicking an outbound thread switches it to the current thread in the Route configuration panel. If there are changes to the current thread, then a prompt opens where you can save or discard the changes.

See [Outbound tab](#).

Moving threads and routes

There are several ways to select threads and routes:

- Select the target. For multiple selections, press and hold **Ctrl**. Selected icons are outlined in blue.
- Drag a selection rectangle around the target.
Any thread/route/reply route that is completely located in the selection rectangle is selected when you release the mouse during the dragging. The thread/route/reply route is not selected even if it is partially located inside the selection rectangle. The dragging rectangle closes immediately after the mouse is released, and all items are shown as selected.

Clear by clicking anywhere outside the selected items.

Repositioning threads

Click a thread icon to select it. Hold and click to drag it to a new location.

You can also do this for multiple threads using **Ctrl + click**.

You can use the arrow keys on the keyboard to move the threads.

Route representation

Highlight the route path and reply route path by clicking the path line.

Route paths use a solid line and reply route paths use a dashed line.

Right-click the route/reply route line to open a menu where you can delete the route. This also opens the Transaction Summary tab.

In Network Monitor, right-clicking a route line and selecting **Transaction Summary** opens the **Transaction Summary** dialog box.

Viewing routes and threads

In a complex configuration, threads and routes fall into sets or subsets of the entire configuration. Often, when network information requires an update, only one of the sets or subsets is affected.

Use the View menu option to show or hide the route path lines in the configuration.

- **Show All Routes**
- **Show All Reply Routes**

The current view file is saved when you save the Netconfig file. This ensures that you do not lose updated view changes when NetConfig is saved.

- There are separate asterisks (*) to mark the modified NetConfig and view files. You can still modify the NetConfig and view files and save them separately.

- There is only one lock alert when a file lock conflict happens during opening/saving Network Configurator.
- If you break the lock during opening a NetConfig file, then you own both the NetConfig and current view file locks. Otherwise, if you do not break an existing lock, both the NetConfig and current view files can be opened in read-only mode.
- If a lock conflict happens when simultaneously saving the NetConfig file and current view, then there is only one alert to inform you about the conflict. You cannot break only the NetConfig lock and open a read-only view or open a read-only NetConfig file and break the view file lock.
- When NetConfig is open and another user attempts to access NetConfig, a message opens stating the NetConfig and view file are locked.

View file location

The `mvw` file is used to store Network Configurator and Network Monitor views.

- This file stores multiple views.
- By default, there is initially no view file in a site's view folder. The IDE attempts to load an existing `dflt.mvw` file when the Network Configurator/Network Monitor are opened.
- If there is no existing `dflt.mvw` in the view folder, then the IDE creates a new `dflt.mvw` file.

In the Network Configurator, both the NetConfig file name and the current view file name are shown in the IDE title.

- The name of the currently opened `mvw` file is appended to the title. The default is `dflt.mvw`.
- The original `*.view` and `monitor_dflt.mvw` files that were created by previous versions can also be loaded.

View List pane

A view list pane is located on the left side of the layout. This displays similar to the view pane in the Network Monitor. The view pane is a horizontally re-sizable pane in Network Configurator and Network Monitor.

The view pane lists all views in the current `mvw` file in sorted order. These views can only be single selected.

The selected view is shown in the layout pane.

By default, the `all_threads` view is automatically selected.

Types of views

There are several types of views in the view pane:

- **Default:**
The `all_threads` view includes the process views and group views. This view contains all threads in the Network Configurator.
Default views are refreshed based on the NetConfig file when Network Configurator/Network Monitor is opened.

NetConfig can be changed separately from the view file. The views in the old view file may not be consistent with the new NetConfig file, especially for default created views. The refresh is performed to make sure the default created views are up-to-date.

- User-created:
 - *process* view
The IDE creates a process view for each process. The default is `process_helloworld`. The process view name is named with the syntax, `process_process name`.
 - *group* view
The IDE also creates a group view for every group and is named with the syntax `group_group name`.

User-created views let you view threads/processes/groups, and can contain the sub-view which references the other views.

When a thread is created in a group or process view, by default the group/process of the created thread is the currently selected group/process.

For the `all_threads` view and user-created views in the Network Configurator, the default process name of a new thread is the site name. The default group name is blank.

You can change the process/group of a thread in a selected view in Network Configurator. If the new process/group does not belong to the selected view, then there is a warning message after you click OK to commit the change: "The new process/group of the thread `thread_name` does not belong to current view. The `thread_name` is removed from this view."

After dismissing the warning message, the changed thread is removed from the current view and it is added to the view to which it belongs.

Adding and editing views

In a complex configuration, it is advisable to view and edit only a certain portion of the entire configuration at one time.

Right-clicking in the View List pane opens a context menu. Selecting **Add New View** or **Edit View** opens the **Create View** dialog box.

Select the view from the menu at the top of the dialog box. This lists all existing views.

If you remove a thread that belongs to the selected view, then the removed thread is added to the source thread list.

If you remove a thread that does not belong to the selected view, then it is removed only from the Threads and Views list.

View filter

Selecting **Filter View** from the right-click menu opens the **Thread Filter Settings** dialog box.

Viewing criteria are ANDed together. For example, if all thread names are listed, and a single group name is listed, only those threads within that group are shown.

On the **Thread Filter Settings** dialog box, you define the view limits to a set of specific processes, group names, or specific threads.

The Filter View accepts names that are partial or case-insensitive.

You can specify the partial name of an object for which you are looking and find that object.

This search is case-insensitive. For example, if the threads are named `phy_beta` and `phy_kappa`, specifying `phy` locates both names.

To see all threads that belong to a single process, but not from a specific group:

- Specify the process name.
- Specify the name of the group to exclude.
- Click **not** after **Group Name**.

The view filter concept applies to individual single views in the `.mvw` file, so that other single views have other filters. You can update filter settings by selecting **Filter View** from the context menu when you right-click in the View List pane.

On this dialog box, you can set the **Process Name**, **Group Name**, and **Thread Name**.

- If the view that is selected is a built-in view, then **Process Name** is disabled. This is because the selected view only contains the threads in the corresponding process.
- If the view that is selected is a built-in group view, then **Group Name** is disabled. This is because the selected view only contains the threads in the corresponding group.
- For the **Process Name** filter, the available process names that can be used for filtering are those whose threads are in the selected view. This applies also to the **Group Name** and **Thread Name** filters.

Filters are not based on the entire NetConfig, but to the sub-set of the NetConfig. This sub-set is restricted by the selected view.

Editing thread filter values

- 1 Right-click a view and select **Filter View**. This opens the **Thread Filter Settings** dialog box.
- 2 Select a view attribute (**Process Name**, **Group Name**, or **Thread Name**).
- 3 Click the corresponding **Choose** button to open the **Filter Values** dialog box for that attribute.
- 4 Use the left and right arrows to place selected values in the left column and unnecessary values in the correct column.
- 5 Click **Apply** when finished.
The values that are listed in the **Filter Values** dialog box are those defined in the network configuration. This is regardless of any view limitation currently in effect.
- 6 Click **OK** on the **Thread Filter Settings** dialog box to apply the view criteria.

HCI_static_else_route

This handles transactions that do not match any configured Static Else or route. This includes any defined Static Else, wildcard Static Else, or `_HCI_static_route_`.

There are two locations that show all the routes that are related to a certain thread: the **Routes Replies** tab and the **Route** dialog box.

Similar to a Static Route, the **Static Else Route** option creates `_HCI_static_else_route_`.

You can add the Static Else Route to any set of routes, where this route is selected if the message fails to determine a valid route. All normal processing continues along this route, whether the transactions are logged and stopped, or translated and processed. This ensures that you know what happens with each transaction.

With this route type, you can keep message counts in sync. This type also logs to the process log and can accept advanced custom auditing.

You can also define specific transactions for ancillary routes, where the remaining transactions must go through this default Static Else route. These show all the routes in other types and the destinations on one page.

Note: `_HCI_static_else_route_` and `_HCI_static_route_` cannot be used together. The engine checks for `_HCI_static_else_route_`, and reports a warning if both `_HCI_static_route_` and `_HCI_static_else_route_` are defined. When this happens, `_HCI_static_else_route_` does not work.

Inter-site routing

Inter-site routing can route to a destination thread that is located in another site and even on another host.

To send to a thread in the current site, you must have the destination thread be a public thread in the target site. You can set a thread to be public by assigning an **Inter-Site Routing Port** to it from the **Properties** tab.

By default, **Inter-Site Route Port** is empty, which means the thread is not a public thread.

After this is set, the thread is a public thread.

You can set a port to make the thread a public thread by:

- Specifying the port in the field.
or
- Clicking **Select** and selecting from the **Select Port** dialog box.

Note: It is your responsibility to ensure the port is not occupied by other threads or applications.

Inter-site routing ports can be configured with the service name and the port number. The Service Name/Port Number pair must be configured exactly the same on the destination side and the side where the process is running.

After the port is set properly, you can commit the inter-site setting successfully with other thread properties.

An icon on the upper-right corner of the thread indicates the referenced public thread.

Creating a destination to represent a public thread from an outside site

A destination, which represents a public thread from an outside site, can be created in the current site.

Creating a new destination automatically populates the **Properties** tab.

- 1 Click **Configure Thread Routes** in the Network Configurator layout panel.
- 2 Drag and drop the destination thread onto the graph panel. You can also right-click in the panel and select **New Destination**.
This is enabled on all_threads view, built-in group views, and user-defined views.
This option is disabled if the destination thread from an outside site does not belong to any process in the current site.
- 3 Accept the default name or specify a new **Destination Name**. The default name is automatically created with dest_1. A number is appended to ensure there is no name duplication in the current site. This name can be changed.
The reply message routing's transaction ID is the source thread name in the message. The destination name for reply routing can be different from the source thread name because these two definitions come from different contexts. To have reply inter-site routing, keep the destination name the same as the source thread name of the outbound message that requests the reply.
- 4 Add a **Bitmap** and **Group Names** are optional. If a bitmap setting is not specified, then the default icon is used to represent the destination thread. You can use any bitmap for the destination. All other fields are required.
- 5 Using the **Host** menu, select the target host IP or host name. Click **OK**, or press **Enter** after selecting a host name or IP address. The IDE starts to connect to the host and attempts to download the site list.
If the target host has security enabled, then a log-in dialog box opens to let you log in. After successfully connecting to the target host, the host is added to the **Host** menu.
When the IDE is connected to a remote site, and loads a NetConfig file:
 - If NetConfig already stores *hostname*, then the IDE shows the host server name, or IP address, when the **Destination Configuration** dialog box is opened.
 - If you specify *hostname* in the field, then a prompt opens to confirm if this is the connected host server or the client machine. It also confirms whether to always save *hostname* to the machine IP on which the IDE is running, not the server machine IP.
- 6 For **Site**, all site names are downloaded from the currently connected host and added to the **Site** menu. The current site on the current host is not on the **Site** menu. There is no requirement to do inter-site routing to a thread in the current site.
The first site in the list is automatically selected. You can select a different site as required.
After any site is selected or the selected site is changed, the thread is updated to list all the public threads in the selected sites. The thread can be empty if there is no public thread in the selected site.
- 7 **Port** is automatically populated with the port read from the public thread if the thread is selected from an outside site.
- 8 Click **Refresh** to update the site list, public thread list, and their port numbers from the selected host. When the refresh has successfully finished, the menu lists and the **Port** text are automatically updated with the newest data.
Note: The **Host**, **Site**, **Thread**, and **Port** fields are all editable. You can directly specify values without connecting to the target host.

9 Click **Verify Thread**.

Only the basic validation is performed to ensure that **Destination Name**, **Host**, **Site**, **Thread**, and **Port** are not empty. The committing is blocked if any of the necessary fields are empty.

To check whether the configured host, site, thread, and port are valid, clicking **Verify Thread** does the whole validation for the destination thread. In addition to checking whether the values are empty, it also attempts to connect to the specified host server and open the specified site. When the site is opened, it checks if the public thread exists on the site and that the port is matched with the current one. If any of these are invalid, then the dialog box automatically goes to the **Verifications** tab with a list of all errors.

Using msgmetaset and msgmetaget

msgDestThread(msg) can hold external destination configuration, similar to:

```
hcitcl>msgmetaset message0 DESTCONN destName:destHost:1020
hcitcl>msgmetaget message0 DESTCONN
destName:destHost:1020
```

When assigning the values to messages by the msgmetaset command, destName, destHost, and destPort are the minimum values set.

You must ensure the correctness of these values.

Inter-site destination menu

Putting the focus on the Destination thread populates the Properties panel.

Right-click an inter-site destination icon to open a menu of options.

The inter-site destination thread works as a thread holder. It can be copied, pasted, and deleted similar to other threads. This is the same as the copy, paste, and delete behavior of other threads, except that the target, and the content, is the destination thread. The pasted destination thread name follows the name rule of the thread, which appends a number to ensure no duplicated destinations are created.

For example, you can copy a destination and paste it into the all_threads view and the corresponding built-in view or any user-created view.

The destination thread cannot be pasted in a Network Configurator process view. It can, though, be pasted in a group view as it does not belong to any process in the current site.

Note: The **Copy/Cut/Paste** actions can only be performed on Network Configurator. These actions are not supported on Network Monitor.

Routing to a destination thread

After creating and configuring a destination thread, you can then route to the destination thread. The destination thread is listed on the thread list for the route destination, similar to any other thread in the current site.

Clicking **List** for the **Destination** field on the **Route Details** dialog box opens the **Threads** dialog box.

After committing the selected destination thread as a route destination, the route line from the source thread to the destination thread is shown.

On the destination site, if there is any destination on the source site referencing a public thread, an icon is shown on the public thread. This indicates the referenced public thread.

This indicator only displays after synchronization is performed from the source site.

Configuring a reply route with inter-site routing

If you must configure a reply route when using inter-site routing to have the reply work across the sites, then:

- Configure the source as a destination available to other sites, which includes defining an externally available port number.
- Add the original source as a destination in the target site.
- Ensure the names match between the destination in the target site and the source in the original source site.

Synchronizing inter-site routing

The public thread can have reference to it or not, even though the properties (for example, port) of the destination thread may be out-of-date.

Inter-site synchronization is a way to synchronize between the source and destination sites. Inter-site synchronization ensures that the destination site knows which public thread has reference from the source site. The source site updates the destination thread's properties to ensure it is current.

Note: During inter-site synchronization, the host server in which the destination site is located must be running.

To synchronize between the source and destination sites, select **File > Inter-Site Synchronize**.

- The inter-site synchronization action is based on the saved NetConfig file. If any changes are made to the current file, then you must save it before you can do an inter-site synchronization.
- If the destination site's host server has security enabled, then a log-in dialog box opens to let you log in.

Note: The IDE instance that is configuring the source site and doing the synchronization must have the user certificate for the connecting destination host server.

After the inter-site synchronization starts, a dialog box opens to indicate the progress and successful completion.

Errors/exceptions during synchronization prompt an error dialog box, stating which host server failed.

Host list

The Network Configurator maintains the host list that contains all of the destination thread's hosts, including the deleted destination thread's host. During synchronization, the source IDE attempts to connect to the target host.

- The source IDE, which is the current IDE doing the synchronization, uploads the referenced public thread list to the target host. For the host, there is no destination to it in the current site, so the list can be empty. The entry is still uploaded.
- The destination host server updates the referenced list. It drops files for any added destination and removes threads that no longer exist.
- The destination host server notifies the connected IDE to update the dialog box for the referenced public threads. This is based on the updated referenced list that shows referenced public threads.
The host server notifies the registered IDE to update reference indicators in the dialog box after any reference information is changed through the host server. For example, after deleting the reference information file during synchronization.
- The current IDE then knows which host has been successfully synchronized. For the successful host, if there is no longer any destination reference to it, it is removed from the maintained host list.

The host list is maintained in a separate data file from the NetConfig file in the source site. The reference information is maintained with a file in the destination site.

The NetConfig on the source site can be changed during inter-site synchronization if any destination port is updated.

After the NetConfig file is changed, a modification mark displays on the NetConfig file name.

Inter-site destination in Network Monitor

The destination in the source site works similar to the common thread, except that the icon can be the default destination icon. You can change it to any available icon, even the common thread icon.

The destination thread has no protocol status icon and no status message. It only shows the thread icon and name.

The destination thread has a menu when you right-click the icon.

- **Notes** opens Notes for a destination. This is enabled only if notes are attached to the destination.
- **Create View** creates a view on the fly. You can also select **View > Create View**, after highlighting the icon.

Note: The **Copy/Cut/Paste** actions can only be performed on Network Configurator. These actions are not supported on Network Monitor.

Wild card routing example

A wild card route can route a set of messages to one destination. For example, all ADT messages to one destination would use a "ADT.*" wild card.

When a route is first created, a **Create Message Route** dialog box automatically opens.

After the route is created, double-clicking the route/reply route name on the Property panel opens the **Route** dialog box.

Consider a route configured to use wild card routing using one of these patterns:

- ORU_R01_2.(3.1)|4.*

- `ORU_R01_2\.(3\.1)|4.*`
- `ORU_R01_2\\. (3\\.1)|4.*`
- `ORU_R01_2.{3.1,4}*` (this is a valid glob pattern, not a valid regex)

Messages using one of these patterns are rejected in state 101 with this error:

```
[xlt :rout:ERR /0: SUBMIT_xlate:02/21/2008 11:35:58] No routes defined for TrxId
'ORU_R01_2.4_19460407STEF02'
```

The last expression gives this error:

```
Bad Regular Expression Route 'ORU_R01_2.{3.1,4}*' couldn't compile regular expression pattern:
invalid repetition count(s)
```

When this TrxID is tested in `hcitcl`, all above variants match:

```
hcitcl>regexp -- {ORU_R01_2.(3.1)|4.*} ORU_R01_2.4_19460407STEF02 1
hcitcl>regexp -- {ORU_R01_2\.(3\.1)|4.*} ORU_R01_2.4_19460407STEF02 1
hcitcl>regexp -- {ORU_R01_2\\. (3\\.1)|4.*} ORU_R01_2.4_19460407STEF02 1
```

The only way to make this work is to remove the parentheses, in which case you must split up the expression into two routes:

```
ORU_R01_2.4.*
ORU_R01_2.3.1.*
```

You must also add the `".*"` to the end.

This regular expression works:

```
{^ORU_R01_2\.( (3\.1) |4) . * }
```

This message was used with a MSH similar to:

```
MSH|^~\&|D|EH|PE|XYZ|20070214011500||ORU^R01_2.4_CSZIS|401150112333744|P|2.5|||A
```

In this example, you must use HL7 version 2.4 or higher. Earlier versions are limited to 7 bytes for MSH.9.

Note: In Tcl, `".*"` at the end is not necessary, whereas in NetConfig it is.

Route details

On **Route Messages** tab of the layout panel, double-click a route type to open the **Route Details** dialog box, where you select the detail type. An "X" on the icon represents an error. Click the **Verifications** tab for details.

You can also click **Configure Thread Routes** and drag a detail type from the toolbar and drop it onto a transaction ID node. The **Details** tab content depends on the translation type: Raw, Xlate, Generate, XSLT, Chain, or Branch.

Configuring a raw route detail

On the **Route Messages** or **Route Replies** tab, double-clicking the Type opens the **Route Details** dialog box.

- 1 Click **Edit** next to **Procs** to open the **TPS Editor**.
- 2 On the **TPS Editor**, click **Add** to select the procedures to run before message transmission. This opens the **TPS Properties** dialog box.
Select **Proc** or **Java Class** to specify a Tcl procedure or Java Class to run, and type any arguments.
Select/clear **Only display procs at local site** to hide/show global/master procs to make the procedure list smaller.
- 3 Click **OK** to return to the **Route Details** dialog box.
- 4 On the **Route Details** dialog box, click **List** next to **Destination**.
This opens the **Threads** dialog box, where you can select the destination thread.
- 5 Select **Enabled** to enable the route.
Clearing the check box disables the route.
- 6 Click **OK** on the **Route Details** dialog box to apply the selections.

Configuring an xlate route detail

On the **Route Messages** or **Route Replies** tab, double-clicking the Type opens the **Route Details** dialog box.

- 1 Click **List** next to the **Destination**.
This opens the **Threads** dialog box.
- 2 Select the destination thread and click **Apply**.
- 3 Click **Edit** next to the **Pre Procs**.
This opens the **TPS Editor**.
- 4 Click **Add** to open the **TPS Properties** dialog box, where you select the procedures or Java class and arguments.
On this dialog box, you can select the language for the TPS script.
- 5 Specify any arguments to add and click **OK** to close the **TPS Properties** dialog box and return to the **TPS Editor**.
- 6 Click **OK** to return to the **Route Details** dialog box.
- 7 On the **Route Details** dialog box, click **List** next to the **Xlate**.
This opens the **Xlate list** dialog box. Xlates must be defined to make a selection.
- 8 Select the translation specification file that contains the translation actions for this route and click **Apply**.
- 9 Click **Edit** next to the **Post Procs**.
This opens the **TPS Editor**.
- 10 Click **Add** to open the **TPS Properties** dialog box, where you select the procedures or Java class and arguments to run after message translation.
- 11 Specify any arguments and click **OK** to close the **TPS Properties** dialog box and return to the **TPS Editor**.
- 12 Click **OK** to return to the **Route Details** dialog box.

- 13 Select **Read-Only** to make this route read-only.
- 14 To elevate warnings to errors when parsing incoming HL7, UN/Edifact, NCPDP, or X12 messages, select **Elevate**.
- 15 Select **Enabled** to enable the route. Clearing the check box disables the route.
- 16 Click **OK** to apply the selections.

Configuring a generate route detail

On the **Route Messages** or **Route Replies** tab, double-clicking the Type opens the **Route Details** dialog box.

Note: The Generate route detail is similar to Raw, except that there is only the **Procs** option.

The generate route detail only controls a message route, using a single user-written Tcl procedure. It does not usually modify the source message or directly change its metadata, although it has that option.

The input for a generate procedure is a keyed list of remaining route details and the source message. The input to a generate procedure is a keyed list of remaining route details. Completed route details are not included in the list.

The generate detail can:

- Modify the input detail list. For example, it can insert new details into the list or remove existing details.
- Add additional generate details to the route detail list that are called when processed.

- 1 Click **Edit** next to the **Procs**.

This opens the **TPS Editor**.

- 2 Click **Add** to select the procedures to run before message transmission.

This opens the **TPS Properties** dialog box.

- 3 Select **Proc** or **Java Class** to specify a Tcl procedure or Java class to run.

- 4 Specify any arguments.

- 5 Click **OK**.

This returns you to the **TPS Editor**.

- 6 Click **OK** on the **TPS Editor** to close it.

- 7 Select **Enabled** to enable the route. Clearing the check box disables the route.

- 8 Click **OK** on the **Route Details** dialog box.

Configuring an XSLT route detail

On the **Route Messages** or **Route Replies** tab, double-clicking the Type opens the **Route Details** dialog box.

- 1 Click **List** next to the **Destination**.

This opens the **Threads** dialog box.

- 2 Select the destination thread and click **Apply**.

- 3 Click **Edit** next to the **Pre Procs**.

This opens the **TPS Editor**.

- 4 Click **Add** to open the **TPS Properties** dialog box, where you select the procedures or Java class and arguments.

On this dialog box, you can select the language for the TPS script.

- 5 Specify any arguments to add.
- 6 Click **OK** to close the **TPS Properties** dialog box and return to the TPS Editor.
- 7 Click **OK** to return to the **Route Details** dialog box.
- 8 Click next to the **XSLT**.

This opens the **XSLT** list dialog box.

This dialog box lists all XSLT files that are located in the `$HCISITEDIR/xslt` and `$HCIMASTERSITEDIR/xslt` directories. Root-level XSLT files are shown in blue italics.

XSLT validation files are supported for HL7 CDA 2.0 and HITSP_C32 (2.5 version). These files are located in the root level XSLT folder (`$HCIROOT/xslt`). They can be used by `hcixslttest` or the engine without any changes. They can also be copied and modified to the site level for special use. This is at `$HCISITEDIR/xslt` or `$HCIMASTERSITEDIR/xslt`.

- 9 Select the `xsl` file for this route.
- 10 Click **Apply**.
- 11 Click **Edit** next to the **Post Procs**.
This opens the **TPS Editor**.
- 12 Click **Add** to open the **TPS Properties** dialog box, from where you can select the procedures or Java class and arguments to run after message translation.
- 13 Specify any arguments and click **OK** to close the **TPS Properties** dialog box and return to the **TPS Editor**.
- 14 Select **Enabled** to enable the route.
Clearing the check box disables the route.
- 15 Click **OK** to apply the selections.

Configuring a Chain route detail

On the **Route Messages** or **Route Replies** tab, double-clicking the Type opens the **Route Details** dialog box.

Note: Previous NetConfigs can be read by versions 5.8 and later. The current version of NetConfig that contains threaded or Chain xlate configuration information is not compatible with previous versions.

The Chain type is shown as "chain first_chain_item | second_chain_item" on the Properties panel.

You can view a message when it goes through each Chain detail. For example, you can see the message output after it goes through an `xlt` file in the Chain. Then, you can see the message after it goes through another `xlt` file.

The output that is displayed is each xlate's output in the route testing for the Chain or Branch.

Three route types are available for mixed Chains:

- **Raw** (default)
Tells the engine to pass the message data through without translation. This can also contain procs to be run before message transmission.

- **Xlate**

Contains Pre Procs, Xlate, Post Procs, Read-only, and Elevate Warnings to Errors. These behaviors are the same as the Xlate Route details, translating the message according to the actions from a previously configured translation file. This is the typical translation mechanism.

The configuration specifies the name of a translation specification (XLT) file, generated by the Translation Configurator, to control the translation process.

- **XSLT**

Contains Pre Procs, Xlate, and Post Procs. These behaviors are the same as the XSLT Route details. XML messages are taken as inputs and translated into other formats according to the formats specified in the XSLT file.

Clicking **List** opens a dialog box that shows all XSLT files located in the %HCISITEDIR%\xslt and %HCIMASTER SITEDIR%\xslt directories. Root-level XSLT files are shown in blue italics.

- 1 For **Destination**, click **List** to open the **Threads** dialog box, where you select the destination for the Chain routes. You can also select an inter-site destination (a thread within another site under the current root).
- 2 Click **Add** to open the **Chain Route Details Properties** dialog box. Newly added Raw/Xlate/XSLT names are appended and listed after the last highlighted item.
For highlighted items, click **Edit** to open the **Chain Route Details Properties** dialog box where you can edit as necessary. The sequence of the items are kept the same as before even if the Raw/Xlate/XSLT names are modified.
Arrange the sequence of the list items by clicking **Move Up** or **Move Down**, or click **Delete** to remove a destination. **Move Up** and **Move Down** are disabled if there is no item or only one item in the list box.
- 3 Select **Enabled** to enable the route. Clearing the check box disables the route.
- 4 Click **OK** to apply the Chain Raw/Xlate/XSLT setting.
- 5 Click **Cancel** to cancel any updates or changes and return to the previous state.

Configuring a Branch route detail

On the **Route Messages** or **Route Replies** tab, double-clicking the Type opens the **Route Details** dialog box.

This dialog box shows the properties of the selected node that is the common part of a Branch translation. The common part is a Chain translation but it has no destination. You can add/delete chain items as required.

There must be at least one Chain and Branch detail, otherwise an error message results.

You can view a message when it goes through each chain detail. For example, you can see the message output after it goes through an xlt file in the chain. Then, you can see the message after it goes through another xlt file.

The output that is displayed is each xlate's output in the route testing for the chain or Branch.

Chained details

Click **Add** to configure Chained details. This opens the **Chained Route Detail Properties** dialog box, where you can select a Type and Properties for that type.

In translation chains:

- The output from a translation is the input to another translation.
- The links in these chains are routing details of the form used for each transaction ID.
- Each link of the chain has the same destination specified for the detail.
- Because a chain with one link is the same as the translation detail in use, chains with a single link are not permitted.

Branch details

Click **Add** to configure Branch details. This opens the **Branch Route Details** dialog box, where you can select a Type and Properties for that type.

The Branch detail has one common translation and its children which are Branch items.

After a Branch item is selected on the tree, the view updates with the new Branch.

Right-click the Branch node to open a menu of editing choices.

Copy/Paste actions can be made across different levels, so that a sub-Branch can be copied and pasted as its parent's sibling.

The common/trunk and Branch chains can have a single translation detail as its only link.

Branch chains cover a common problem where a message must be normalized and then passed to several translations by TCP threads:

- The source thread xlate fixes the message for later processing.
- With only one translation per destination the normalized translation is repeated for each outbound destination thread.
- The outbound thread then uses TCP to send the message to another thread where the normalized message is processed.
- Branch chains reduce this overhead.
- The number of protocol threads and their overhead is reduced.
- The chain that does the normalization is the common, or trunk, chain.

Characteristics of a basic Branch chain:

- Consists of one common/trunk translation detail that does the normalization.
- The output of this translation is then passed to one or more translation details that are the Branches.
- Each Branch has a unique protocol thread destination that receives the result of the Branch translation.
- Second level Branches are not supported. There is only one Branch point in a Branch chain.
- Branches are restricted in that they must each have a unique specified destination thread. One of the Branch's destination must be used for the translation details of the common/trunk chain.
- There is no restriction on the number of single details, chains, or Branch chains that are run under a single TrxID. Each may have from zero to as many as are required.

At this point, you can:

- **Add Route Detail After**
- **Add Route Detail Before**
- **Add Branch Detail**
- **Copy/Cut/Delete** the selected Branch item. After doing a **Copy/Cut**, you can paste a Branch item to the Route detail level or Branch detail level.

A new Branch item becomes a Branch child to the Branch detail.

Enabling the route

After configuring the Chained and Branch details, select **Enabled** to enable the route. Clearing the check box disables the route.

Configuring a reply route

Similar to route configuration, this panel improves the usability of configuring reply routes and related configurations.

Reply routes can be configured to destinations other than the source thread.

Note: For route replies, a static route only processes the messages which cannot find a matched defined TrxID in the route replies configuration.

These translation types are available:

- Raw
- Xlate
- XSLT
- Chain

- 1** Click **Configure Thread Routes** to open the configuration panel.
- 2** Click **Configure Reply Routes**.
- 3** Configure as required. Dotted lines connect the nodes.
- 4** Click **Configure NetConfig** to return to the Network Configuration main panel.

Reply thread properties configuration

This panel is shown when you select a reply thread node in the Route Graph panel.

See:

- [Inbound Replies pane](#)
- [Verifications tab](#)

Reply route trxID configuration

On the **Route Replies** tab, click **New Route** to open the **Route** dialog box.

This table shows the available configuration items:

Parameter	Description
Name	<p>The transaction ID indicates the type of message. For example, what type of transaction is represented by the record. The route name must match the Trx ID in the record, including case. Specify a transaction ID name. The existing route names from the same source thread are listed in the . If there is no existing route name, then the is empty. You can specify a new route name or select any existing one.</p> <p>Specify a transaction ID name. The existing route names from the sameFor message-type record formats, specify the name of the message type.</p> <p>For fixed-length record layouts, specify the exact characters that display in the transaction ID field of the inbound data message.</p>
Static route	<p>A static route processes all messages the same way, regardless of their transaction ID. Use a static route to apply a common action to all transactions. For example, sending a copy of all messages to an archival site.</p> <p>When you select an existing route name, the current route belongs to the selected route name.</p> <p>For Static else route, see HCI_static_else_route.</p>

Parameter	Description
Wild Card Route	<p>This setting can be shared by multiple routes. If the shared setting is changed, then a warning message opens after you click OK: "The changed Wild Card Route is shared by the routes: raw1, raw2. This change will be applied to all of them."</p> <p>Select this option to route a set of messages to one destination. For example, all ADT messages to one destination would use a wildcard: ADT.*.</p> <p>The Name behaves as a regular expression. For a message to be considered a match, its transaction ID has to be a complete match to this expression.</p> <p>For example, the wildcard ADT_A0\d is the equivalent of the Tcl regexp <code>{^ADT_A0\d\$} \$transactionID</code>. This wildcard matches a message whose transaction ID is ADT_A01, but does not match a message whose transaction ID is ADT_A17 or ADT_A01a.</p> <p>If negative lookahead is used in the wildcard route, then the expression must be properly written. For example, to match a route started with ADT_A and it is not ended in 17 with negative lookahead expression, ADT_A(?!17)\d\d should be used. ADT_A(?!17) does not work.</p> <p>To use regular expression shorthand in wildcard route, double slashes should be given. For example, to make the above negative lookahead expression work as wildcard route, the correct Name value is ADT_A(?!17)\\d\\d.</p>
Remove all messages when any route detail fails	<p>Select this to define route or reply actions when an error happens in a route detail. If the check box is selected, then all messages are dropped. Otherwise, only the error route detail is dropped.</p>
Store original message in metadata for downstream processing	<p>The pre-xlate message is cached in Route Details and is used in the Database-Outbound Protocol Properties dialog box and Database-Outbound protocol engine thread. When this is selected, the engine thread caches the pre-xlate message at runtime. By default, this is cleared.</p>
Enabled	<p>Enables/disables the route.</p>

Reply route details configuration

This tab is for configuring translation properties for the currently selected route type node.

The content depends on the translation type:

- Raw node. See [Configuring a raw route detail](#) on page 577.
- Xlate node. See [Configuring an xlate route detail](#) on page 577.
- XSLT node. See [Configuring an XSLT route detail](#) on page 578.
- Chain node. See [Configuring a Chain route detail](#) on page 579.

Add a destination node by dragging a thread from the thread list and dropping it over a route type node.

Outbound reply route configuration

For configuration details, see [Outbound Replies pane](#).

Configuring, customizing, and migrating message archiving

From the Site Preferences **External Database** tab:

- 1 Select a **Database Connection** and select **Unlimited** or specify the maximum number of kilobytes the log file can attain before cycling.
Automatic Log Cycling Threshold automatically cycles message logs when they get to the specified size. Cycling a save file causes the current file to close, renames it to <old_name>.old, and opens a new save file. The saved files are located in the subdirectory named %HCISITEDIR%\exec\hcimsgarchive.
- 2 Configure the thread that uses this feature. On the NetConfig's **Properties** tab, select **Message Archiving** and click **Setup**. The archiving alerts are internal and MonitorD loads them automatically.
- 3 Set up the alert by entering a **Table Name**. Then, accept the default for the timing and frequency for the message to be written to the external database, or change as required. If the **Table Name** is new, a message opens to confirm generating the new table.

Setup Message Archiving dialog box

Note: The message archiving feature is licensed as an add-on module.

If message archiving functionality is not used, then the engine and daemon behavior are unchanged. Earlier NetConfig files can be loaded, but after they are saved by the new dialog box, they cannot be loaded by previous versions.

Clicking **Setup** next to **Message Archiving** opens the **Setup Message Archiving** dialog box. In this dialog box, you can specify an alert configuration file to contain the alerts used to write the messages to the external database.

Parameters	Description
Table Name	This is the name of the table in the external database. Multiple threads can be written to the same table. Cloverleaf supports letters, digits, and underscores in table names and column names.
Message Count Threshold	The number that is entered is the threshold for writing messages to the database. The default is 15.
Timeout in Seconds	The time-out threshold for alerts that are generated. The default is 10 minutes, or 600 seconds.
Proc	The Tcl procedure to modify the message data. If this is not specified, then the message is processed as it should.
Only display procs at local site	Select/clear this option at local site to hide/show global/master procs to make the procedure list smaller

When the setup message archiving is configured and **OK** is clicked, a dialog box opens showing the command result.

Additional information

For information on message archiving configuration, migrating from previous versions, and command line usage, see

- [Configuring, customizing, and migrating message archiving.](#)
- [hcimsgarchive.](#)

Customizing the message archiving table

Located in `$HCIR00T/archiving` are two SQL files that you can customize for message archiving.

- `create_template.sql`
This is used to create the table.
In this file, `TABLERNAME` is replaced by the Message Archiving table configured for the thread.
- `insert_template.sql`
This is used to insert the data into the table.
In this file, `TABLERNAME` is replaced by the Message Archiving table configured for the thread.
The other fields in angle brackets are replaced by the matching metadata field when the message is stored.

By modifying these files, you can control the table structure, data types, and what metadata fields are stored. It is suggested that you put any modifications into `$HCISITEDIR/archiving`.

Migrating from previous versions

When migrating from a previous version's message archive to version 6.2 and later:

- Before doing the migration, all messages in the database cache must be archived into the external database.
- Use the version 6.2 and later database. The previous version's database cannot be used as the database version has changed.
- Configure message archiving for each thread. The internal alert takes effect according to the configuration.

Remove the previous version's external alerts. Although the previous version's external alerts can work in version 6.2 and later, it is best to remove them. Version 6.0 and later uses internal alerts.

Examples of using Network Configurator

These examples show how to use Network Configurator to configure a communication network, use auto-start, and use the recovery database.

Configuring a communication network

- 1 Add two thread icons to the Network Configurator:
 - **his_1a** as the data source
 - **lab_1a** as the destination for the data
- 2 For his_1a, add a new route and set it as static. This configures the static route with a his_1a to lab_1a raw route detail.
- 3 On the **Properties** tab, set up the his_1a thread protocol properties to use your input file. To do this, on the **Properties** tab click **Properties** next to Protocol. For this example, an input file named `frl_adt_.dat` is used.
- 4 Configure the transaction to isolate the Trx ID from the first four bytes of the record on his_1a and lab_1a threads. Do this by clicking **Edit** next to **Trx ID Determination Format**.
- 5 On the **Route Messages** tab, double-click the route name to open the **Route** dialog box. Edit the his_1a static route by highlighting the route name and replacing `_HCI_static_route_` with `ADMT`.
- 6 On the **Properties** tab, designate all threads as part of the group named `production_ex` and clear **Auto-start connection**.
- 7 On the Inbound pane of the **Properties** tab, for his_1a, select **Save inbound messages**. Name the file `inx_bound`. This is part of the SMAT saved message file. Do not add a suffix to this file name.
- 8 Highlight the lab_1a thread and on the **Properties** tab configure the protocol properties to write out to a file named `his_1a.out`.
- 9 Highlight the his_1a thread and click **Verify Thread** or right-click the thread icon to open the context menu and click **Verify**. Any errors are reported on the **Verifications** tab.
- 10 Click **Apply** and save the file.
- 11 Save the views by selecting **View > Save Views**.

- 12 Testing your routes can speed up the overall verification of your interfaces. To do this, open the Testing Tool and select the **Routes** tab.
 - a Select **his_1a** as the source thread.
 - b Select the data file (in this example, `frl_adt_.dat`).
 - c For **Send to Proc**, select **hci routetestshowbydest**.

Using auto-start and the recovery database

- 1 Drag and drop the File protocol icon onto the Graph panel to create two threads named **thread1** and **thread2** with a group name of **production_ex2**.
- 2 For thread1, add a new route to thread2 and set it as static when the **Create Route** dialog box opens.
- 3 Click **Apply** and save the file.
- 4 On the thread1 **Properties** tab, select **Save inbound messages** in the Inbound pane and specify **thread1_in** in **File**. Leave **Auto-start connection** and **Use recovery database** selected in the Thread pane.
- 5 Click Protocols **Properties** to open the protocol dialog box. Browse and select the **Input Filename** and click **OK**. In this example, it is `frl_adt_.dat`.
- 6 Click **Apply** and save the file.
- 7 Highlight thread2. In the Properties Outbound pane, select **Save outbound messages** and specify **thread2_out** in **File**. In the Thread pane, **Auto-start connection** and **Use Recovery Database** are selected by default.
- 8 Click **Apply** and save the file.
- 9 Open the Network Monitor and run the process *process*. In this example, it is `trgprod`. Because **Auto-start connection** is selected by default, both thread1 and thread2 start.
- 10 Shut down the process.
- 11 Open the *process* folder under `%HCIR00T%\process name\exec\processes\process name`. There are two files: `thread1_in.msg` (saved inbound message) and `thread2_out.msg` (saved outbound message).
- 12 Return to Network Configurator. Highlight thread2 and clear **Auto-start connection** on the **Properties** tab. Click **Apply** and save the file.
- 13 Open Network Monitor and run the *process name* process. thread2 should not be started.
- 14 Open the Database Administrator and click **Apply Command**.
- 15 Click **Done** to close the result pane. Then, shut down the process.

Saving the configuration to a file

It is best to save the configuration periodically when working on it, so that no work is lost. Save the configuration view, which defines how the symbols display in the Network Configurator.

Some changes to the network configuration affect the network configuration file database. Others affect only the view of the configuration as it displays in the Network Configurator.

For example, moving an icon in the dialog box affects only the view and not the configuration database itself.

In general, there is only one configuration file per site, but any number of views of that configuration can be created and saved.

- To save the network configuration to a file, select **File > Save** in the Network Configurator. The save operation verifies the configuration before saving it. The configuration is automatically saved to the file named NetConfig (unless saved to a different file using the **File > Save As** option).
- To save the network view to a file, select **View > Save View** in the Network Configurator. The default view file name is `config_dflt.view`, unless another file name is specified using the **View > Save View As** option.

Protocols

When computers communicate with one another, a set of rules, known as a protocol, is used to transmit and receive data in an orderly fashion. Protocols are the meaning and sequencing rules for requests and responses used for managing a network, transferring data, and synchronizing the states of network components.

Most protocols are similar in their configuration elements:

- Several protocols contain the **Auto-Reconnect** and **Reopen Time** options.
Auto-Reconnect specifies if the connection to the host should reopen if it goes down. Select this to specify the reopen time between retries to open the connection to the host.
- All protocols contain **Start-Up Procs**.
This specifies the names of Tcl procedures to run when the thread starts. This is a Tcl Procedure Stream (TPS).
When a thread starts, the startup TPS runs in Startup mode. There is no message passed into it. Thereafter, access to this TPS is controlled by the Protocol Startup Switch.

Length-encoding

The TCP/IP protocol uses length-encoding. Message length-encoding provides a way to determine message boundaries, letting the driver know how many characters are in the message. After the driver decodes a string, it reads exactly that many subsequent characters and processes them as a message.

Advanced scheduling

Advanced Scheduling uses the Network Configurator to specify schedules in terms of absolute dates and time, instead of offset seconds. This includes Fileset Local, Fileset FTP, HTTP Client, and UPoC .

IPv6

Internet Protocol is a set of technical rules that defines how computers communicate over a network. There are currently two versions: IP version 4 (IPv4) and IP version 6 (IPv6).

IPv6 (Internet Protocol version 6) is the next generation protocol designed by the IETF (Internet Engineering Task Force). This replaces the current version Internet Protocol, IP Version 4 (IPv4). IPv6 is expected to gradually replace IPv4, with the two coexisting for many years during a transition period.

IPv4 versus IPv6

IPv4 was the first version of Internet Protocol to be widely used, and still accounts for most of today's Internet traffic.

IPv6 is a replacement for IPv4. IPv6 was deployed in 1999 and provides far more IP addresses, which should meet the requirement well into the future.

Most of today's Internet uses IPv4. There is a growing shortage of IPv4 addresses, which are required by all new machines added to the Internet.

IPv6 fixes many problems in IPv4, such as the limited number of available IPv4 addresses. It also adds many improvements to IPv4 in areas such as extended address space, auto-configuration, and others.

The major difference between IPv4 and IPv6 is the number of IP addresses. There are over 4 billion IPv4 addresses. In contrast, there are over 16 billion-billion IPv6 addresses. The technical functioning of the Internet remains the same in both versions. Most networks that use IPv6 support both IPv4 and IPv6 addresses in their networks.

In the system, two sockets are created: one for IPv4 and one for IPv6, both of which are handled separately.

Scope

There are three types of scope for the IPv6 unicast addresses:

- The link-local scope identifies all hosts within a single layer 2 domain. With this, two hosts on separate networks could have the same link-local address. For example, if host B is required to communicate with host A, the scoped IPv6 address "fe80::1%eth0" must be used.
- The site-local scope fits logically between the link-local and global scopes. A site can be a corporate network, a geographically constrained portion of a corporate network, or an administered domain within a corporate network. A site-local address is unique in the site.
- The global scope identifies all devices reachable across the Internet. Global unicast addresses are unique.

Scalability

When **IPv4/v6 Dualmode** is enabled and the server is bound to both 0.0.0.0 and ::, one extra listening socket is created for IPv6. **Local Binding Address** is blank.

Two sockets are created, one for IPv4 and one for IPv6, both of which are handled separately.

- If the server is multi-server and the number of maximum concurrent connections to this server is N, then N+2 sockets are allocated. One for listening, one extra IPv6 listening port, and N for the N connections on the port.
- If a server is single, then three sockets are allocated. One for listening, one for the connection, and one extra IPv6 listening port.

In summary:

- If only one IPv4 socket is required, then clear **IPv4/v6 Dual**. Selecting **IPv4/v6 Dual** and setting **Local Binding Address** to "0.0.0.0" or other valid IPv4 address do the same.
- If only one IPv6 socket is required, then select **IPv4/v6 Dual**, and set **Local Binding Address** to ":::" or other valid IPv6 address.
- If **IPv4/v6 Dual** is selected and **Local Binding Address** is blank, then two listening sockets are created on the server.

Protocol Properties dialog boxes

IPv6 is supported in Fileset FTP/FTPS/SFTP, HTTP/HTTPS Client, TCP/IP, and PDL TCP/IP.

On these dialog boxes, there are fields named **Host** and **Local Binding Address**.

In previous versions, when only IPv4 was supported, the **Host** field was used to specify the server's host name or IPv4 address on the client side. The **Local Binding Address** field was used for local binding on both client and server side.

With IPv6, these fields also accept IPv6 addresses, using one of these formats:

- Host name:
 - *hostname*
 - *www.sitename.com*
 - and so on.
- IPv4 address:
 - "127.0.0.1" (loopback address)
 - "192.168.0.5" (private network address)
 - "64.233.167.104" (global network address)
 - and so on.
- IPv6 address:
 - "::1" (loopback address)
 - "fe80::210:c6ff:fe95:a586%4" (link-local address)
 - "fe80::5efe:192.168.123.1%2" (link-local ISATAP address)
 - "3ffe:501:ffff::1234" (global unicast address)
 - and so on.

See <http://www.ietf.org/rfc/rfc4007.txt>.

Local Binding Address can be one of these strings:

- "0.0.0.0" (IPv4 wildcard address)
- ":::" or "0:0:0:0:0:0:0:0" (IPv6 wildcard address)
- Left blank (":::" and "0.0.0.0")

The longest field is the IPv6 address that includes an embedded IPv4 address, and a scope identifier appended after a "%" character:

```
xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:ddd.ddd.ddd.ddd%scope_id
```

In this example:

- Each "x" is a hexadecimal digit, "0" through "9" and "A/a" through "F/f."
- Each "d" is a decimal digit, "0" through "9."

For link-local addresses:

- `scope_id` can be a numeric value standing for the interface index, or a string as the interface name.
- `scope_id` length is system-dependent. On Windows, it is at most 10. The total length of such an IPv6 address is no more than 56 characters long.

URL field in HTTP Client Protocol Properties dialog boxes

In the **HTTP Client Protocol Properties** dialog box, there is also a **URL** field.

An HTTP URL is used to ask the HTTP client to perform an HTTP request.

For example, `http://www.sitename.com/index.html`.

In this case, you use `GET /index.html` to the host `www.sitename.com` at the default HTTP port 80.

The **Host Name** can be replaced by an IPv4 address.

For example, `http://64.233.167.104/` is the same as `http://www.sitename.com/`.

If a port other than the default is used, then it can be given in the URL field after the host name or IP address with a ":" character as delimiter.

For example, `http://www.sitename.com:8080/` OR `http://64.233.167.104:8080/`.

When an IPv6 address is used in the URL field, it must be enclosed in brackets.

For example, `http://[::1]/` and `http://[::1]:8080/` are valid URLs. It also states that this syntax does not support IPv6 scope ID.

Windows can support scope ID in their WinINET interface. It is required that the "%" character be percent-encoded.

For example, the IPv6 address "`FE80::2%3`" must display in the URL as `http://[FE80::2%253]/`.

"%25" is the percent-encoding of the "%" character.

IPv4/v6 Dual check box in TCP/IP and PDL TCP/IP Protocol Properties dialog boxes

On the TCP/IP and PDL TCP/IP, there is a **IPv4/v6 Dual** check box.

This check box is provided for backward compatibility. The default status is cleared.

- When the box is cleared, there is only one IPv4 listening socket that is created on the server. Only one IPv4 socket is used for the connection to server on the client.
- When the box is selected and **Local Binding Address** is left blank, two listening sockets are created to accept incoming connections on the server. One for IPv4 and the other for IPv6. On the client, one kind of socket is used to connect the server. If it fails, then another one is used. The sequence, IPv4 or IPv6 first, may vary on different operating systems.

Database protocols

The Database protocol supports VRL messages in addition to the current database schema format. The Database Outbound protocol can query data from the database and return the data as a reply message. The **Inbound (Outbound) SQL Statement Configurator** dialog box supports all tables in the database. The Table Configuration tool supports calling stored procedures.

The Database protocol also supports custom statements once the database connection is made. Windows Authentication is supported natively when connecting to SQL Server.

Additional support is provided for:

- Multiple tables in the same SQL schema.
- Role assignment to connections after connection.
- Auto trim for white space in fields.
- BLOB and CLOB

See [BLOB and CLOB](#) on page 366.

For error handling, meaningful error messages are logged in the engine log file. Meaningful, human, and machine readable errors are returned to the user for action by UPoC scripts or other means.

All configurations are stored at the site level.

This table describes the different database protocol functions:

Function	Description
Database Outbound Protocol: VRL	The Database Outbound protocol supports the use of VRLs as the outbound staging message format. You can modify the VRL structure after it has been created.
Database Outbound Protocol: Select statement support	The Database Outbound protocol supports using the outbound staging message to query the database. The query result set is returned as a reply message.
SQL Configuration: All database table support	All tables in the database are supported for select, insert, delete, and update statements.
Lookup Table: Stored Procedure support	Dynamically stored procedures are supported in the Lookup Table Configurator. You can edit the stored procedures to assign input and output variables. Return values and result sets are mapped for use by the engine.
Database Configuration: Connect Success	You can define SQL statements that are run upon a successful database connection.

Connect Success action

Double quotes around the SQL Connection Success action (`connect_success_action`) are not required.

Connect Success Action does not support these DML/DQL:

- create
- drop
- insert
- update
- delete
- select

Note: Doctrine Query Language (DQL) is an Object Query Language created for helping users in complex object retrieval. A data manipulation language (DML) is a computer programming language used for adding (inserting), deleting, and modifying (updating) data in a database.

Database protocols user interface

The **Inbound (Outbound) SQL Statement Configurator** is composed of Options and SQL panels.

Clicking **Select** on the **Inbound (Outbound) SQL Statement Configurator** enables **Table/View**, **Column**, and **Condition**.

The **Table Selection** dialog box contains a list of all tables in the current database connection.

- Clicking **Select** beside **Column** opens the **Select Column** dialog box. This lists all the column names of the currently selected table/view.
- Clicking **Configure** beside **Condition** displays the **Configure Condition** dialog box.
The Outbound staging **Fields** list box contains all the field addresses in the currently selected Outbound Staging Message format.

In the Options panel, the Table/View Schema list displays all the column names in the selected table/view as defined in the **Inbound (Outbound) SQL Statement Configurator** dialog box.

Double-clicking or selecting an item in the Outbound Staging Message fields list or Table/View Columns list appends the item to the cursor position in the Condition text area. The name is enclosed with angle brackets.

Database Outbound protocol: VRL

You can use the **Database Outbound Protocol** dialog box to select VRL or Table Schema as the data format. This also applies to the **Inbound (Outbound) SQL Statement Configurator** dialog box.

The **Stored Procedure Configurator** dialog box is used to select VRL or Table Schema as the data format. The Columns list supports both VRL field names and Database columns.

SQL Configuration: All database tables support

The **Inbound (Outbound) SQL Statement Configurator** dialog box can access all tables and views in the current database connection.

Lookup Table: Stored Procedure support

In the **Lookup Table Configurator > Advanced Database Lookup** configuration, you can make free text edits to the database statements. **SQL** and **Stored Procedure** are used to indicate the type of database statement. Clicking **Stored Procedure** enables the **Stored Procedure Out Parameters** dialog box.

This dialog box contains check boxes that you can use to define the expected return from the stored procedure:

- **Return Code**
- **Result Set**
- **Out Parameters**

One of these must be selected. All are selected by default.

Database Configuration: Connect success

You can manually enter SQL statements to be run upon a successful connection. Global variable substitution is supported for the SQL statements. If the SQL fails to successfully run, then the thread enters an error state. Errors are logged and added to the protocol error field in thread statistics.

The Database protocol Testing Tool is supported.

Options panel

Schemas are automatically filled when the **Inbound (Outbound) SQL Statement Configurator** is opened. You can configure the schemas in this dialog box or when you make a query from the configured connection.

On the Column/Value panel, you can select the **Column** and **Value** to use.

- **Column** indicates the full column name in [table/view name].[column name] format.
- **Value** is editable. The default value is the corresponding value of **Column** enclosed with angle brackets. For example, <table1.column1>.

When selecting an item on the Table/View Schema list, all related column items are automatically appended to the table and all rows are checked by default.

When an item is cleared from the Table/View Schema list, the related columns are also removed from the table.

When double-clicking a row in the table, the corresponding column's full name is appended to the SQL text area.

From the toolbar, you can **Move Up** or **Move Down** as required, to change the table row sequence.

Condition panel

Select **Configure** to compose the condition clauses in the text area.

Type panel

This panel is used to select the SQL statement type. You can **Insert**, **Update**, **Delete**, or **Select**.

- When **Insert**

Only table-type schemas can be selected. You cannot use multiple schema selections.

The standard SQL statement is composed. For example:

```
insert into <checked table> (<checked column list>) values (<checked value list>)
```

This becomes:

```
insert into table1 (table1.column1, table1.column2) values (<table1.column1>, <table1.column1>)
```

Generate>delete from <checked table> where <condition string> is enabled when the checked column list is not empty.

- When **Update** is selected, the behavior of the Table/View schema list is identical to the is selected, the Condition panel does not display. Generate SQL **Insert** type.

The standard SQL statement is composed. For example:

```
update <selected table> set < checked column/value list> where < condition string>
```

The selected column/value list is composed of column [0] = value [0], column [1] = value [1]

For example:

```
update table1 set table1.column1=<table1.column1>, table1.column2=<table1.column2>
where table1.column1=value1 and table1.column2=value2
```

Generate SQL is enabled when the checked column list is not empty.

- When **Delete** is selected, the Column/Value table does not display.
- Only table-type schemas can be selected. You cannot use multiple schema selections.

The standard SQL statement is composed. For example:

```
Generate>delete from <checked table> where <condition string>
```

This becomes:

```
delete from table1 where table1.column1=value1 and table1.column2=value2
```

Generate SQL is enabled when the checked table list is not empty.

- When **Select** is selected, column/value table label changes to Column. The Value column does not display.
- The standard SQL statement is composed. For example:

```
select <checked column list> from <checked tables/view list> where < condition string>
```

This becomes:

```
select table1.column1, table1.column2, table2.column3 from table1, table2
where table1.column1=value1 and table1.column2=value3
```

Generate SQL is enabled when the checked column list is not empty.

Message Panel

This panel is only used for the Database Outbound protocol.

The message format tree is generated by the outbound staging option that is configured in the **Database Outbound Protocol Properties** dialog box. When you double-click a message format tree node, the value that is enclosed with angle brackets is set to the Value cell of the selected row in the Column/Value table. If no component is selected, then it is appended to the SQL text area.

When **Whole Message** is selected, the `<whole_message>` string is set to the Value cell of the selected row in the Column/Value table. If no component is selected, then it is appended to the SQL text area.

SQL panel

The SQL text area is editable. This is where the generated SQL statement displays.

If content is defined, then the string displays in the SQL text area when reopening the dialog box.

On the Table/View Schema list, double-clicking a schema appends the corresponding name to the SQL text area.

Select **Generate SQL** to generate a SQL statement using the **Schema**, **Column**, and **Condition** options. This is disabled by default.

This populates the generated SQL statement in the SQL text area.

Example: VRL

To build a VRL-to-database interface for storing spreadsheet data in a database:

- 1 Create an inbound thread to read the `csv` files.
- 2 In the **Database Outbound Protocol Properties** dialog box, select the **Database Connection** named “accounting” and a VRL named “finance”.
- 3 Open the Network Configurator and create a new thread named “db_outbound”.
- 4 Select `database-outbound` as the protocol and click **Properties**.
- 5 Select the `accounting` database connection and the “finance” VRL as the outbound staging message format.

Example: Outbound select statement

To build an interface that queries a database for patient documents from incoming HL7 messages, you must query the database by the patient MRN and Visit Number.

To do this:

- 1 Configure a **Database Connection** named “documents” and create a new thread named “check_docs”.
- 2 Select `database-outbound` as the protocol and click **Properties**.
- 3 Select the `documents` database connection and click **Configure** to open the **SQL Configurator** dialog box.
- 4 Click **Select** and configure the fields to be returned as a reply message.

After the interface is deployed, the MRN and Visit number are extracted from the inbound transactions. The “check_docs” thread queries the database with the MRN and Visit number. Any documents returned in the result set are returned to the engine as a reply message.

Example: Lookup Table support

To use multiple stored procedures in an interface:

- 1 Open the Lookup Table Configurator and select **Advanced Database Lookup**.
- 2 Select the **Database Connection**.
- 3 Click **Configure** to open the **Inbound (Outbound) SQL Statement Configurator** dialog box.
- 4 Select the stored procedure to configure and assign the In and Out variables.
- 5 Click **OK** to save the configuration.
- 6 Repeat the process for the other stored procedures that are required to dynamically lookup.

Example: Connect Success

In the Connect Success SQL panel, you can specify SQL statements to run upon a successful connection. Cloverleaf Global Variable substitution is supported for the SQL statements. If the SQL fails to successfully run, then the thread enters an error state. Errors are logged and added to the protocol error field in thread statistics.

The default value is empty.

The Database Protocol test tool is supported.

For example, to assign role permissions to a database connection:

- 1 Open **Site Preferences > Database Configurations**.
- 2 Select the **Database Connection** and specify the role permissions in the Connect Success SQL panel.

When the interface is deployed and started, the SQL statements are immediately run when the database connection is made.

Configuring a database inbound protocol

In the Java Driver and Database protocols, one process only supports one Java or database protocol. They cannot be in the same process. In the Java Driver, you can also run multiple instances in a single process.

Table Schema lists all selected visible table schemas under the selected connection. Selecting a different **Database Connection** clears the **Table Schema** field.

Note:

- This option can be left blank. Table schema is not limited to one of the table schemas in the dialog box. You can insert/delete/update any tables in the database. All database tables are supported for the insert/delete/update statement on the **Inbound SQL Statement Configurator** dialog box.
- If a database connection from the master/current site has no visible schema, then it is not shown on the Database Connection menu.

- When using escape characters, the defined escape characters should not display in the message as normal content. If the characters have real meaning in the message, then you can define other characters as escape characters. The engine can handle them as normal content.
- 1** Select a **Database Connection**. This is required. This lists all configured visible connections under the current site and master site. Master site connections are shown in italic.
- 2** (Optional) Click the **Table Schema** button to open the **Select Table Schema** dialog box.
 - a Click **Add** to open a list of available table schemas. Multiple selections are permitted.
 - b Click **OK** to populate the **Table Schema** field with the selections.
- 3** The Action tabs are used to configure the action of the database inbound protocol. Select **SQL Statement** or **Stored Procedure**. These are used to identify the action type. The **Content** field shows the action content. Selecting **SQL Statement** updates **Content** with the previous SQL statement content. Clicking **Configure** opens the **Inbound SQL Statement Configurator** dialog box, from which you configure the SQL statement. By default, **SQL Statement** is selected.

See [SQL configuration](#).

Selecting **Stored Procedure** is used mainly for selecting data with complex query conditions and deleting rows after reading. This updates **Content** with the previous stored procedure content. Click **Configure** to open the **Stored Procedure Configurator** dialog box, from which you configure the statement that calls the stored procedure.

See [Stored procedure configuration](#).

For the **Read Success Action** and **Read Failure Action** tabs, you select **SQL Statement** or **Stored Procedure**. These two actions are optional for inbound configuration.

The **Read Success Action** tab defines the post action for read success.

The **Read Failure Action** tab defines the post action for read failure.

- 4** In the Scope pane, select **Each row as a message** or **Each read as a message**.
Each row as a message identifies each row in the query result as a message. This is the default.
Each read as a message identifies all entries in the query result as one message.
- 5** In the Max Row per Read pane, select the maximum row number of query results or **Unlimited**.
Unlimited means the query result involves all data rows.
 Or, you can set the maximum row number of query results by selecting several rows to be read. The default value is 1. This reads one row at a time. Selecting any other number reads that number of rows. If you specify an uncertified DBMS that does not support the max row setting, then **Max Row Per Read** must be set to **Unlimited**.
- 6** Specify a **Scan Interval**. This is used to define a fixed timing interval in which to scan the database to retrieve data for each interval. This value is an integer greater than zero that represents seconds. The default value is 30.
 The Scheduling pane is used to define the timing interval at which the current thread scans the database. Scanning the database means that the thread runs the query statement of read action to query data from the database. The database is scanned based on the defined timing interval. In each scan, the queried row count is defined by the **Max Row per Read** setting.
- 7** If required, then select **Use advanced scheduling**.
 Selecting this enables **Setup** and **Connect as needed** and disables **Scan Interval**. This means the current thread follows advanced scheduling to scan the database. By default, **Use advanced scheduling** is cleared.

Clicking **Setup** opens the **Scheduling** dialog box.

- 8 Select **Connect as needed** for the current thread to create a temporary connection to run the database scan when the advanced scheduling time is up. Otherwise, the current thread creates a persistent connection.

If **Use advanced scheduling** is selected, then by default **Connect as needed** is also selected.

Clearing **Use advanced scheduling** disables **Setup** and **Connect as needed** and enables **Scan Interval**. In this case, the current thread always creates a persistence connection to run the database scan.

Configuring a database outbound protocol

You can use these steps to configure the SQL statement/stored procedure for the database outbound protocol.

In the Java driver and database protocols, one process only supports one Java or database protocol. They cannot be in the same process. In the Java driver, you can also run multiple instances in a single process.

- 1 On the **Database Outbound Protocol Properties** dialog box, select a **Database Connection** (required). This lists all configured visible connections under the current site and master site. Master site connections are shown in italic.
- 2 Select **Close connection after write** or **Delay connection until needed**. Both can also be selected.
Close connection after write closes the connection after every outbound write. Clear this to keep the connection.
Delay connection until needed delays the connection until it is required. Clear this to make the connection during thread start-up.
- 3 Click the Message Format **Select** button to open the **Select Record Format** dialog box. This lists all selected visible table schemas under the selected connection. Make a selection and click **OK**.
Sometimes, the format of the message coming in does not match any table schema, or, the table schema is unknown. If this happens, then you can use the **VRL** format option to define a VRL for messages without having to match any database table schema.
To clear the text field, select **Clear**.
- 4 For Action Options type, select **SQL Statement** or **Stored Procedure**.
Selecting **SQL Statement** updates **Content** with the previous SQL statement content. Clicking **Configure** opens the **Outbound SQL Statement Configurator** dialog box, from which you configure the SQL statement.
See [SQL configuration](#).
Selecting **Stored Procedure** is used mainly for selecting data with complex query conditions and deleting rows after reading. This updates **Content** with the previous stored procedure content. Click **Configure** to open the **Stored Procedure Configurator** dialog box, from which you configure the statement that calls the stored procedure.
- 5 Select **Use cached pre-xlate message as whole message** to define whether the current thread uses the cached pre-xlate message or the outbound message as the whole message. When this is selected, the Database Outbound protocol driver attempts to extract the cached pre-xlate message from the message driver control. Then, it decodes it if it is waiting to be used.

whole_message placeholder

The SQL statement action supports a *whole_message* placeholder that represents a whole message.

In the **Database Outbound Protocol Properties** dialog box, there is a **Use cached pre-xlate message as whole message** check box. This is used to define whether the current thread uses the cached pre-xlate message or the outbound message as the whole message.

By default, the check box is cleared. When selected, the Database Outbound protocol driver attempts to extract the cached pre-xlate message from the message driver control. Then, it decodes it with base64 decoding, if waiting to be used.

Use the *whole_message* placeholder to insert/update a table column with the cached pre-xlate message. After this, the engine replaces the *whole_message* placeholder with the retrieved pre-xlate message if it exists at runtime.

You can directly specify the *whole_message* placeholder into the SQL statement action. You can also click **Configure** to open the **Outbound SQL Statement Configurator** dialog box in which to compose a SQL statement with the placeholder.

Outbound SQL Statement Configurator dialog box

In this dialog box, clicking **Select** next to the **Column** field opens the **Select Column** dialog box. The table component lists all the selected columns, and the **Whole Message** check box represents whether the column must be inserted/updated with a whole message.

For the selected columns, their values are replaced with *whole_message*. For example:

```
INSERT INTO callback(callback.THREADNAME, callback.PROCESS)
VALUES (whole_message, callback.PROCESS)
```

In the **Outbound SQL Statement Configurator** dialog box, after opening the **Select Column** dialog box, clicking **Add** opens the **Column List** dialog box. This uses a table component to display columns. After making a selection, clicking **Apply** closes the **Column List** dialog box and applies the selected columns to the **Select Column** dialog box. If required, then at this point you can change the check box status on the **Select Column** dialog box.

Network Configurator

In the **Route Messages** or **Route Replies** tab of Network Configurator, double-clicking a route name opens the **Route** dialog box.

For the **Store original message in metadata for downstream processing** option, the pre-xlate message is cached in route details. This is used in the **Database Outbound Protocol Properties** dialog box and database outbound protocol engine thread.

When this option is selected, the engine thread caches the pre-xlate message at runtime. By default, this is cleared. The engine's xlate thread encodes the pre-xlate message. Then, it caches it into the current message's metadata driver control field before sending it to the destination thread. After this, the destination thread that is defined in the **Route Details** dialog box can use the cached message.

Inserting a whole message using the `whole_message` placeholder

The SQL statement action supports a `whole_message` placeholder that represents a whole message.

- 1 Open the **Site Preferences** dialog box and on the **Database Configurations** tab configure a database connection. Or, open a previously created database connection.
- 2 Define the database schema format to use in the Database Outbound protocol. To do this:
 - a Open the **Database Schema Configurator**.
 - b Import table schema by clicking the **Import** button.
 - c Click **OK**.
- 3 In the Network Configurator **Database Outbound Protocol Properties** dialog box, click the **Table Schema** button and select a table schema from the **Select Table Schema** dialog box.
- 4 Select **SQL Statement** option and click **Configure**.
- 5 Select a **Table/View**.
Click **Select** to open the **Table Selection** dialog box. Select from the list of all tables in the current database connection.
- 6 For **Column**, click **Select** to select one or more columns for the selected table/view.
- 7 After selecting a column, click **Whole Message** for the selected columns.
Use the `whole_message` placeholder to insert/update a table column with the cached pre-xlate message. After this, the engine replaces the `whole_message` placeholder with the retrieved pre-xlate message if it exists at runtime.
- 8 If required, then click the **Configure** for the **Condition** field.
See [Configuring a condition](#).
- 9 Click **OK** to close the **Outbound SQL Statement Configurator** dialog box.
This inserts `whole_message` into the Value pane.
For example:

```
INSERT INTO callback(callback.PROCESS,
callback.CONTEXT) VALUES (<callback.PROCESS>, whole_message)
```

- 10 If required, then select the **Use cached pre-xlate message as whole message** check box. This defines whether the current thread uses the cached pre-xlate message or the outbound message as the whole message.
By default, the check box is cleared. When selected, the Database Outbound protocol driver attempts to extract the cached pre-xlate message from the message driver control. Then it decodes it in base64 decoding, if it is waiting to be used.
- 11 In the **Route Messages** or **Route Replies** tab of Network Configurator, double-click a route name to open the **Route** dialog box.
Select the **Store original message in metadata for downstream processing** option. The pre-xlate message is cached in route details and is used in the **Database Outbound Protocol Properties** dialog box and database outbound protocol engine thread.
When this option is selected, the engine thread caches the pre-xlate message at runtime. By default, this is cleared. The engine's xlate thread encodes the pre-xlate message. Then, it caches it into the current

message's metadata driver control field before sending it to the destination thread. Then, the destination thread that is defined in the **Route Details** dialog box can use the cached message.

12 Send messages to the Database Outbound protocol using the Testing Tool.

Note: The Database Outbound protocol parses a message that is based on the defined table schema. To insert the entire message to one single column, the message must not include the field separator and termination characters.

Inserting a whole message using a Tcl UPoC

To insert a whole message without using the *whole_message* placeholder, you can do this using a Tcl UPoC.

- 1** Define an appropriate table schema including all the columns to be used in the SQL statement.
- 2** Write a Tcl script to generate a new message that matches the defined Table Schema format based on the original message.
- 3** Apply the Tcl script in the pre-outbound TPS.
- 4** Write an appropriate SQL statement to do the insert/update.

Insert an HL7 message to a single column

Generally, HL7 messages do not contain the default field separator and termination of Table Schema format. In this case, an HL7 message can be inserted into a single column.

Inserting a VRL/database schema message into a single column

In the Database Outbound protocol, you can use a VRL as the outbound staging message format. In the dialog box, you can select Variable Length Record (VRL) or Table Schema as the data format.

You can modify the VRL structure after it has been created.

In the **Outbound SQL Statement Configurator** dialog box, you can select Variable Length Record (VRL) or Table Schema as the data format.

In the **Stored Procedure Configurator** dialog box, you can select Variable Length Record (VRL) or Table Schema as the data format. The Column list supports VRL field names and database columns.

There are two methods for inserting a VRL/database schema message:

- Method #1: Enclose the whole message in an escape pair (double-quote). You can write a Tcl script to enclose the whole message in double-quotes to escape the field separator/termination.
- Method #2: Change the field separator/termination of the Table Schema format used in the Database Outbound protocol to make it different from the one in incoming messages.

To do this, from the Database Schema Configurator, select **Options > Table Schema Options**.

Change the field separator to a different character. For example, you can change it to a semicolon (;). Now, the comma character in the original message is not a field separator anymore. The entire message is parsed as one field and inserted into a single column.

dbp module

For debug purposes, use the dbp EO module in the Engine Output Configurator. This shows you the full SQL statement being passed on to the driver and run against the database.

By enabling this module or enabling all engine output, the database protocol runtime information is shown in the engine log.

Inserting an HL7 message example

This is an example of inserting an HL7 message and its patient ID into Table2's columns HL7content and patientID.

- 1 Create Table2, including the columns patientID and HL7content, where patientID is the primary key.
- 2 Click the **Table Schema** button and import Table2 for the Database Outbound protocol.
- 3 Write a Tcl script to generate a new message:

```
$patientID $fieldSeparator $originalMessage
```

- \$patientID is retrieving from the \$originalMessage.
- \$fieldSeparator is the field separator defined in Table Schema format. Field separators can conflict with the message content, so you can set an appropriate field separator in Table Schema to avoid conflicts.
- \$originalMessage is the raw message.

- 4 Apply the Tcl script to the pre-outbound TPS.
- 5 Run insert sql statement:

```
INSERT INTO Table2(Table2.patientID, Table2.HL7content)
VALUES (Table2.patientID,Table2.HL7content)
```

- 6 Run update sql statement:

```
UPDATE Table2 set Table2.HL7content=Table2.HL7content
WHERE Table2.patientID=Table2.patientID
```

Dynamic SQL

On the database outbound protocol UI, you can select to use **Use Dynamic SQL**.

“dynamic_sql” is a placeholder for the database outbound protocol. This is used as the content of an outbound action. The inbound message is treated as a SQL statement by the database driver if the outbound action content is set to be this placeholder.

This placeholder is replaced by the inbound message, which is a special SQL statement, that is run by the JDBC driver. In other words, with this placeholder you can dynamically customize the SQL statement that is within a message.

When the placeholder is used, there is no requirement to create an outbound staging message format to match with the database table schema.

Example:

The database driver reads a row from `table_a`:

```
mykey c1 c2 isread
-----
big test1 test2 0
```

These actions are within the message:

- `DELETE * FROM table_a WHERE mykey='big';`
- `UPDATE table_a set isread=1 WHERE c1='test1' and c2='test2';`
- `INSERT INTO table_a(mvkey,c1,c2,isread) VALUES('big','test1','test2',0);`
- `SELECT * FROM table_a WHERE mykey='big';`

The placeholder is flexible, so that it can introduce SQL injection.

For security considerations, the outbound protocol database driver limits the functionality. The action must belong to the SQL DML approval list, which consists only of `INSERT/UPDATE/DELETE/SELECT`. The database driver verifies the action before running.

The jar package that includes the dynamic SQL functionality is saved as a java contrib file, using the name `CJDDDBProtocol-DynamicSQL.jar`, residing in `$HCIR00T/lib/java/contrib`.

BLOB/CLOB support for uncertified databases

Note: See the Release Notes for the latest supported BLOB/CLOB databases.

Other types of databases are supported if they are up to the JDBC standard. These include `mysql`, `PostgreSQL`, and others.

Because different databases might have different definitions of the BLOB/CLOB datatype, the database protocol driver cannot handle all of the various data types. To support these additional databases, CIS has a customized `largeObjectType` connection property for the database URL. With this property, CIS can specify a JDBC that handles BLOB/CLOB for uncertified databases. The value of `largeObjectType` must be one of `oracle` (default), `sqlserver`, or `sqlite`.

LargeObjectType Value	Expectation
<code>oracle</code>	BLOB/CLOB is treated as Oracle database.
<code>sqlserver</code>	BLOB/CLOB is treated as MS SQL SERVER database.
<code>sqlite</code>	BLOB/CLOB is treated as SQLite database.

For example, if a MySQL database is taken, the `largeObjectType` property in database JDBC URL is set as `oracle`:

```
{DB_URLjdbc:mysql://192.168.1.10:3306/cloverleaf_test;largeObjectType=oracle}.
```

`largeObjectType=oracle` indicates the database protocol driver deals with the BLOB/CLOB data type in MySQL in the same manner as Oracle.

Additional information is available at:

<https://docs.microsoft.com/en-us/sql/connect/jdbc/setting-the-connection-properties?view=sql-server-ver15>

Database Outbound reply generation

You can use the reply generation feature in a Database Outbound protocol thread to returned results. These results are from a SQL statement or stored procedure that are to be returned to the engine on the reply stack. These replies can be routed to other connections/threads for further processing.

The Database Outbound protocol uses **Await replies** to determine if it must generate a reply message. This option is on the **Outbound** tab. The reply message is based on the return results of SQL statement/stored procedures.

- If **Await replies** is selected, then a reply message is generated. The returned results from the database invocation is passed back to the engine as a reply message.
- If **Await replies** is cleared, then a reply message is not generated.

Reply generation of a SQL statement

The Database Outbound protocol driver retrieves all results of a SQL statement. The **Update Count** and **Results Set Count** are saved into the message metadata's `DRIVERCTL` field. Result sets are saved into message content.

For example:

```
Reply message:
msgDriverControl: {_CLUPDATECOUNT_ 1} {_CLRSCOUNT_ 0}
message: ''
```

- `{_CLUPDATECOUNT_ 1}` indicates 1 row is updated in the outbound SQL statement.
- `{_CLRSCOUNT_ 0}` indicates no result set is returned from the outbound SQL statement.

Reply generation of a SQL stored procedure

In addition to the returned **Update Count** and **Results Set Count** objects, two other types of results can be returned from a stored procedure. These result types are return code and OUT parameter values.

Similar to the SQL statement reply generation logic, the Database Outbound driver retrieves all the results of a stored procedure.

Only result set objects are saved into message content. The remaining results are saved into `DRIVERCTL` as a keyed list where the key is the stored procedure's name.

For example:

```
Reply message:
msgDriverControl: {sp_dbp_in2out2 {{_CLUPDATECOUNT_ 1} {_CLRSCOUNT_ 1} {_CLRC_ 100} {@output1
test1} {@output2 600}}}}
message: ''
```

- {_CLUPDATECOUNT_ 1} indicates 1 row is updated in the outbound stored procedure sp_dbp_in2out2.
- {_CLRSCOUNT_ 0} indicates no result set is returned from the outbound stored procedure sp_dbp_in2out2.
- {_CLRC_ 100} indicates the returned code of the outbound stored procedure sp_dbp_in2out2 is 100.
- {@output1 test1} indicates the outbound stored procedure sp_dbp_in2out2 returns test1 for output variable @output1.
- {@output2 600} indicates the outbound stored procedure sp_dbp_in2out2 returns 600 for output variable @output2.

Reply generation of a SQL chained stored procedure

Generating a reply message for chained stored procedures is similar to a single stored procedure. The difference is each stored procedure returns its own results, so there are multiple sets of results returned from chained stored procedures.

The returned results set objects are saved into message content; other results are saved into DRIVERCTL as a keyed list where the key is the name of the stored procedure.

For example:

```
msgDriverControl: {sp_dbp_insert {{_CLUPDATECOUNT_ 1} {_CLRSCOUNT_ 0}}}
: {sp_dbp_select {{_CLUPDATECOUNT_ -1} {_CLRSCOUNT_ 1}}}
message: {sp_dbp_ select {
: {{RSMETACOLUMNNAME {
: {ID} {strFLD} {intFLD} {dateFLD} {flag}
: }}
: {RSDATA {
: { {keyNull} {test11} {} {} {100} }
: { {key01} {test1} {1} {2012-10-01 00:00:00.0} {1} }
: { {key02} {test2} {2} {2012-10-02 00:00:00.0} {1} }
: { {key03} {test3} {3} {2012-10-03 00:00:00.0} {1} }
: { {key04} {test4} {4} {2012-10-04 00:00:00.0} {1} }
: { {key05} {test5} {5} {2012-10-05 00:00:00.0} {1} }
: }}}
: }}
```

{sp_dbp_insert {{_CLUPDATECOUNT_ 1} {_CLRSCOUNT_ 0}}} indicates one row is updated in the outbound stored procedure sp_dbp_insert. No result set is returned from stored procedure sp_dbp_insert.

{sp_dbp_select {{_CLUPDATECOUNT_ -1} {_CLRSCOUNT_ 1}}} indicates no updated count is returned from the outbound stored procedure sp_dbp_select. One result set is returned from stored procedure sp_dbp_select. The result set content is saved in the reply message's content.

Outbound database Tcl API

The dbp0bGetResultset Tcl procedure is provided with the Database Outbound protocol. You can use this Tcl proc in the **TPS Inbound Reply** UPoC to handle the returned result set of the database-outbound protocol.

By default, this generates a VRL under `HCISITEDIR/formats/variable/thread_name.vrl` based on the returned result set.

The caller can set `{BUILD_VRL 0}` in TPS **Args** to disable the VRL build.

When creating the VRL file or parsing the result set, the script uses the VRL default field separator and termination.

The caller can set key `FIELD_SEPARATOR`, `TERMINATION` in the TPS **Args** to provide user-defined ones.

When building the VRL definition, the field separator and termination are set with hex format.

If the database-outbound protocol is configured to use a stored procedure, then the caller must set `{SP_FLAG 1}` in TPS **Args**.

Example 1

This example builds the VRL and sets the user-defined field separator and termination.

In **TPS Inbound Reply**, select `dbp0bGetResultset()` and set **Args** as `{FIELD_SEPARATOR |} {TERMINATION \n}`.

Result:

VRL `HCISITEDIR/formats/variable/thread_name.vrl` definition is built.

If the VRL file exists, then the VRL definition is not built again.

The data that is parsed by the proc is similar to:

```
key01|test1|1|2012-10-01 00:00:00.0|1
key02|test2|2|2012-10-02 00:00:00.0|1
...
```

Example 2

This example does not build the VRL, but uses the default field separator and termination.

In **TPS Inbound Reply**, select `dbp0bGetResultset()` and set **Args** as `{BUILD_VRL 0}`.

Result:

No VRL definition is built.

The data that is parsed by the proc is similar to:

```
key01,test1,1,2012-10-01 00:00:00.0,1
```

Example 3

In this example, the result set is returned from a stored procedure.

In **TPS Inbound Reply**, select `dbp0bGetResultset()` and set **Args** as `{SP_FLAG 1} {FIELD_SEPARATOR |} {TERMINATION \n}`.

Result:

VRL `HCISITEDIR/formats/variable/thread_name.vrl` definition is built.

The data that is parsed by the proc is similar to:

key01|test1|1|2012-10-01 00:00:00.0|1 key02|test2|2|2012-10-02 00:00:00.0|1

Database Outbound reply message DRIVERCTL

Sometimes, a stored procedure name or output parameter name includes a period (.) in the stored procedure invocation. The DRIVERCTL of the reply message is normalized to a valid keyed list. In these situations, the period (.) in the keys and sub-keys is replaced with a caret (^).

For example, if the stored procedure invocation in NetConfig is:

```
{?=call dbo.sp_dbp_in2out2(Table_sanity.strFLD, Table_sanity.intFLD,
test.output1 OUT, test.output2 OUT)};
```

The DRIVERCTL of the reply message is:

```
msgDriverControl::{dbo^sp_dbp_in2out2 {{_CLUPDATECOUNT_ -1} {_CLRSCOUNT_ 1} {_CLRC_ 0}
{test^output1 value1} {test^output2 value2}}}
```

Related configurations

Related configurations that are used in reply generation are:

- JDBC drivers
By default, they are installed under \$HCIR00T/clgui/lib.
- JDBC driver configuration
\$HCIR00T/server/database.ini
- DB connection configuration
\$HCISITEDIR/dbconfiguration.ini
- Inbound replies, on the NetConfig **Outbound** tab.
Enable outbound **Await replies**.

Returned message control

A GETRETURNVALUE key is added as a sub-key of OB_ACTION to indicate if the outbound database protocol driver generates a returned message.

```
{ OB_ACTION {
  "" { GETRETURNVALUE 0 }
} }
Key name: GETRETURNVALUE
Range: 0 or 1 (0: disable, 1:enable)
Default: 0
```

This works together with **Await replies** to indicate the returned message type:

- If GETRETURNVALUE is disabled and **Await replies** is disabled, then the engine does not get the return value.

- If GETRETURNVALUE is enabled and **Await replies** is disabled, then the engine generates a DATA message for the return value.
- If GETRETURNVALUE is enabled and **Await replies** is enabled, then the engine generates a REPLY message for the return value.
- If GETRETURNVALUE is disabled and **Await replies** is enabled, then the engine writes a warning in the engine log. A REPLY message is still generated for the return value.

Returned message format

Two optional keys, RETURN_TCL and RETURN_SCHEMA, are added as sub-keys of OB_ACTION to indicate the returned result set format:

```
Key name: RETURN_TCL
Range: 0 or 1 (0: not Tcl list, 1:Tcl list)
Default: 0
Key name: RETURN_SCHEMA
Range: empty or a DB SCHEMA name
Default: empty
```

- If RETURN_TCL is set, then the result set is made as a Tcl list. This is for compatibility for old behavior.
- If RETURN_SCHEMA is empty, then the engine generates the return message based on the result set metadata.
- If RETURN_SCHEMA is set a schema name, then the engine generates the return message based on the given schema.
- If both RETURN_TCL and RETURN_SCHEMA are set, then the engine converts the return message to a Tcl list. The column order is based on the give schema.

Migration considerations

hcirootcopy does special handling to add the GETRETURNVALUE, RETURN_TCL, and RETURN_SCHEMA keys during migration.

- If **Await replies** is enabled in the previous version of NetConfig, then GETRETURNVALUE is added and set to "1" in the new version of NetConfig.
- RETURN_TCL is added and set to "1" (Tcl list) after migration.

Stored procedure configuration

When configuring the Database protocol for the first time, you must select a **Database Connection** and **Table Schema**.

Stored procedures are supported in the Lookup Table Configurator. You can edit the stored procedures to assign input and output variables. Return values and result sets are mapped for use by the engine.

Selecting **Stored Procedure** and clicking **Configure** opens the Stored Procedure Configurator. The columns from the selected table are shown.

Syntax

Note: Stored procedure invocations cannot be configured on a SQLite database.

Invoking statement syntax options:

- `{?= call procedure-name[(arg1,arg2, ...)]}`
- `{call procedure-name[(arg1,arg2, ...)]}`

This is the stored procedure SQL escape syntax for stored procedures to invoke in a standard way for all RDBMS systems.

- Only two types of parameters (IN and OUT) are supported in the database protocol stored procedure. The INOUT parameter is not supported.
- When setting a parameter, use the OUT flag at the end of the parameter name. Parameters without the OUT flag are considered as IN parameters.
- The form `{?= call procedure-name[(arg1,arg2, ...)]}` indicates that a result parameter is set to retrieve the return value of the stored procedure.

The engine uses the `_CLRC_` name for the result parameter. The result parameter is registered as an OUT parameter. Not all databases support the stored procedure return value. You must confirm with the database provider before using it.

- In one stored procedure invocation, OUT parameter names should not be duplicated.

Examples:

```
{?= call sptest_rc(inValue1, inValue2, outPara1 OUT, outPara2 OUT)}
```

This invokes the `sptest_rc` stored procedure. This has two IN parameters and two OUT parameters, and requests the return value of the `sptest_rc` stored procedure.

```
{call sptest(inValue1, inValue2, outPara1 OUT, outPara2 OUT)}
```

This invokes the `sptest` stored procedure, which has two IN parameters and two OUT parameters.

NetConfig migration

Migration is required when migrating from CIS 6.0 to CIS 6.2 and later versions.

Run `hcirootcopy` to do the NetConfig migration for the Database protocol's stored procedure invocation.

If a stored procedure is used in the inbound read action for the Oracle database, then you must manually add the OUT `CURSOR` parameter. This retrieves the `resultset`.

Closing the Stored Procedure Configurator

Clicking **OK** closes the **Stored Procedure Configurator** dialog box and populates the configured statement into the Content area on the main dialog box.

If a stored procedure statement has already been defined, then it displays when you open the **Stored Procedure Configurator** dialog box.

Invoking a stored procedure in the Database Inbound protocol's Read Action

There can be one stored procedure in the inbound Read Action. The stored procedure that is used in the Database Inbound Read Action must return a result set. If no result set is returned, then there is no message generated from the Database Inbound protocol.

No place holder is permitted in the inbound Read Action.

For the Oracle database, to return the resultset you must use the `OUT CURSOR` flag to define an `OUT CURSOR` parameter.

Example:

```
{call sptest_select(rowset OUT CURSOR)}
```

The `rowset` parameter is an `OUT` parameter that represents a `CURSOR`. The engine retrieves the result set from the `OUT` parameter `rowset`.

The definition of the `sptest_select` stored procedure is:

```
create or replace procedure sp_dbp_select(out_var out sys_refcursor)
IS
BEGIN
open out_var for select * from TABLE_SANITY order by TABLE_SANITY.ID;
END;
```

Note: `OUT CURSOR` is only used for the Oracle database. Multiple `CURSOR`s are also supported.

Oracle example

```
{call sp_dbp_read_multiple( out_var1 OUT CURSOR, out_var2 OUT CURSOR)}
```

The `out_var1` and `out_var2` are `OUT` parameters that represent two `CURSOR` elements. The engine retrieves the result sets from these two `OUT` parameters.

The definition of the `sp_dbp_read_multiple` stored procedure is:

```
create or replace procedure sp_dbp_read_multiple(out_var1 out sys_refcursor, out_var2 out
sys_refcursor)
IS
BEGIN
open out_var1 for select * from TABLE_SANITY where TABLE_SANITY.FLAG=0 order by TABLE_SANITY.ID;
open out_var2 for select * from TABLE_SANITY where TABLE_SANITY.FLAG=1 order by TABLE_SANITY.ID;
END sp_dbp_read_multiple;
```

Example of other databases:

```
{?=call sp_dbp_read_multiple()}
```

The definition of this `sp_dbp_read_multiple` stored procedure is:

```
create procedure sp_dbp_read_multiple
AS
BEGIN
select * from Table_sanity where Table_sanity.flag=0 order by Table_sanity.ID
```

```
select * from Table_sanity where Table_sanity.flag=1 order by Table_sanity.ID
END
```

Invoking a chained stored procedures in the Database Inbound protocol's Read Success/Failure Action or Database Outbound protocol's Action

A chained stored procedure invocation is supported in the Database Inbound protocol's Read Success/Failure action and Database Outbound protocol action.

When invoking more than one stored procedure, a semicolon (;) is used to separate the stored procedures.

For example:

```
{? = call sptest(inValue1, inValue2, outPara1 OUT, outPara2 OUT)} ; {call sptest(inValue1, inValue2, outPara1 OUT, outPara2 OUT)};
{call sptest_simple}
```

In a chained stored procedure invocation, the procedure name should not be duplicated.

Two types of place holders are supported in the chained stored procedure:

- The database Schema field place holder
- The OUT parameter place holder

With the OUT parameter place holder, you can use the returned OUT parameter value of the previous stored procedure. This value is used as the input value of the latter stored procedure.

Usage is:

```
<procName.parameterName>
```

For example:

```
{?=call sp1(inValue, outPara1 OUT, outPara2 OUT)}; {call sp2(outPara1 OUT)}; {call sp3(<sp1._CLRC_>,
<sp1.outPara1>, <sp1.outPara2>, <sp2.outPara1>) }
```

In this example, the sp3 stored procedure uses the sp1 output value <sp1._CLRC_>, <sp1.outPara1>, <sp1.outPara2> and sp2 output value <sp2.outPara1> as its input value.

Adding the OUT CURSOR parameter in the stored procedure invocation

If a stored procedure is used in the Inbound Read Action for the Oracle database, you must manually add the OUT CURSOR parameter. This goes in the right position of the stored procedure invocation to retrieve the result set.

For example, when using this stored procedure definition in the Oracle database:

```
create or replace procedure sp_dbp_read(out_var out sys_refcursor)
IS
BEGIN
open out_var for select * from TABLE_SANITY where TABLE_SANITY.FLAG=0 order by TABLE_SANITY.ID;
END;
```

Set the stored procedure initiation in the Inbound Read Action as:

```
{call sp_dbp_read(rowset OUT CURSOR)}
```

Example 1

This example passes parameters from a Cloverleaf message into a stored procedure using the Database Outbound protocol.

For example, a web service that returns several fields must be invoked. Then, these fields must be inserted into the database.

Finally, you must pass the parsed fields as parameters into the Database Outbound protocol; that is, you must bind the fields to the stored procedure parameters.

To do this, use the Database Outbound protocol and pass parameters as the placeholder value defined in the outbound stored procedure invocation. That is, you pass parameters to a stored procedure to insert data into the database.

For additional information on the usage of stored procedures, see the online help at **Configuration > Protocols > Database > Stored Procedure Configurator**.

For invoking stored procedures, use any of these syntaxes:

```
{?= call procedure-name[(arg1,arg2, ...)]}
```

```
{call procedure-name[(arg1,arg2, ...)]}
```

This is a stored procedure SQL escape syntax that invokes stored procedures for all RDBMS.

- Only two types of parameters, IN parameter and OUT parameter, are supported in the Database protocol stored procedure. An INOUT parameter is not supported.
- When setting an OUT parameter, use the OUT flag at the end of the parameter name. Parameters without the OUT flag are considered as IN parameters.
- The form `{?= call procedure name[(arg1,arg2, ...)]}` indicates that a result parameter is set to retrieve the return value of the stored procedure.
- The Cloverleaf engine has a certain name "_CLRC_" for the result parameter. The result parameter is registered as an OUT parameter. Not all databases support the stored procedure return value, so you must first confirm with the database provider before using it.
- In a stored procedure invocation, OUT parameter names should not be duplicated.

This example inserts data into a database by a stored procedure:

```
{call sp_dbp_insert(TABLE_SANITY.ID, TABLE_SANITY.STRFLD,  
TABLE_SANITY.INTFLD, TABLE_SANITY.DATEFLD, TABLE_SANITY.FLAG)};
```

tableName.fieldName is a placeholder of the field value *tableName.fieldName* of the Cloverleaf outbound message.

In the Database Outbound protocol, the Cloverleaf engine first parses the outbound message that is based on the table schema format. This format is defined in protocol properties and replaces the placeholder with the *tableName.fieldName* field value.

Command issued:

```
hcidbprotocoltest -e ASCII -f nl oracle_sp_write C:/cloverleaf/cisversion/integrator/
oracle_dbtest/test/test.txt
```

Command output:

```
Test with MESSAGE 1
key01,test1,1,10/01/12,0
Running DB Write action succeeded.
```

SQL result:

```
{call sp_dbp_insert(key01, test1, 1, 10/01/12, 0)}
```

Example 2

This example calls chained stored procedures using the Database Outbound protocol.

For example, a stored procedure must be called. Then, another stored procedure's return code or output parameter value is required as the input parameter. Finally, one stored procedure's output parameter must be set as another stored procedure's input parameter.

To do this, use the chained stored procedures in the Database Outbound protocol. Then, use the *sp Name.paramName* placeholder to pass the output parameter value of the previous procedure into the input parameter of the next stored procedure.

This example calls two stored procedures:

- Procedure `sp_dbp_test1` returns two out parameters, `@output1` and `@output2`, and the return value `_CLRC_`.
- Procedure `sp_dbp_test2` uses `sp_dbp_test1`'s out parameters `sp_dbp_test1.@output1` and `sp_dbp_test1.@output2`. `sp_dbp_test1`'s return code `sp_dbp_test1._CLRC_` is used as its first three input parameters.

Note: When calling more than one stored procedure, a semicolon (;) is used to separate the stored procedures.

```
{?=call sp_dbp_test1(Table_sanity.strFLD, Table_sanity.intFLD,
@output1 OUT, @output2 OUT)};
{call sp_dbp_test2(sp_dbp_test1.@output1, sp_dbp_test1.@output2,
sp_dbp_test1._CLRC_,Table_sanity.dateFLD, Table_sanity.flag)}
```

Command issued:

```
hcidbprotocoltest -e ASCII -f nl sqlserver_sp_test C:/cloverleaf/cisversion/integrator/
sqlserver_dbtest/test/test.txt
```

Command output:

```
Test with MESSAGE 1
key01,test1,1,10/01/12,0
The output value of <sp_dbp_test1._CLRC_> is: 90
The output value of <sp_dbp_test1.@output1> is: liketest1
The output value of <sp_dbp_test1.@output2> is: 6
Running DB Write action succeeded.
```

SQL result:

```
{_CLRC_ OUT=call sp_dbp_test1(test1, 1, @output1 OUT, @output2 OUT)}
{call sp_dbp_test2(liketest1, 6, 90, 10/01/12, 0)}
```

Returned results:

```
Procedure: sp_dbp_test1
> return value: _CLRC_ = 90
> out parameter: @output1 = liketest1
> out parameter: @output2 = 6
Procedure: sp_dbp_test2
> update count: 1
```

Example 3

This example queries data using a stored procedure in the Database Inbound protocol.

For example, to select all records from database table `Table_sanity`, you can create a stored procedure for data selecting. Then, you can pass this stored procedure in the Database Inbound protocol Read Action.

Note: For the Database Inbound protocol, passing parameters to Read Action for the SQL statement or stored procedure is not supported. For the Oracle database, you define an out parameter with `sys_refcursor` type to return the result set. When passing this stored procedure, the `OUT CURSOR` key should be set in the **Read Action** stored procedure implementation at the end of the cursor out parameter.

Query data from a SQL Server database:

- 1 Define the stored procedure on SQL Server:

```
create procedure sp_dbp_read
AS
select * from Table_sanity where Table_sanity.flag=0 order by Table_sanity.ID
```

- 2 Pass the stored procedure in Read Action using:

```
{call sp_dbp_read()}
```

- 3 Run the `hcidbprotocoltest`:

```
hcidbprotocoltest -r read
```

SQL result:

```
{call sp_dbp_read}
Read 1 message(s) from DB.
key01,test1,1,10/01/12,0
```

Query data from an Oracle database:

1 Define the stored procedure on Oracle:

```
create or replace procedure sp_dbp_read(out_var out sys_refcursor)
IS
BEGIN
open out_var for select * from TABLE_SANITY where TABLE_SANITY.FLAG=0 order by TABLE_SANITY.ID;
END;
```

2 Pass the stored procedure in Read Action using:

```
{call sp_dbp_read(rowset OUT CURSOR)}
```

3 Run the `hcidbprotocoltest`:

```
hcidbprotocoltest -r read
```

SQL result:

```
{call sp_dbp_read(rowset OUT CURSOR)}
Read 10 message(s) from DB.
key01,test1,1,04/01/14,0
```

Example 4

In the `simple_stored_procedure_call_with_parameters` thread, the `GetPatientById` stored procedure is called with:

```
{call GetPatientById(patients.id, fname OUT, lname OUT, dob OUT)}
```

The input parameter is `patients.id` and has a value of 1. This fails.

Testing Tool result:

```
outbound db thread
Test with MESSAGE 1
1
Running DB Write action failed.
```

SQL result:

```
{call GetPatientById(IN Id, OUT fname, OUT lname, OUT dob)}
Oct 31, 2014 11:18:52 AM com.lawson.cloverleaf.dbprotocol.CDBStoredProcedure execDbWrite
SEVERE: boo!
```

Stored procedure:

```
CREATE PROCEDURE `GetPatientById`(IN p_id INT, OUT p_fname CHAR, OUT p_lname CHAR, OUT p_DOB CHAR)
BEGIN
    SELECT fname, lname, DOB
    FROM patients
    WHERE id = p_id;
END
```

In the example above, the stored procedure definition is incorrect, so use:

```
CREATE PROCEDURE `test`.`GetPatientById`(IN p_id INT, OUT p_fname VARCHAR(48), OUT p_lname VARCHAR(48), OUT p_DOB VARCHAR(48))
BEGIN
SELECT fname, lname, DOB into p_fname, p_lname, p_DOB FROM patients WHERE id = p_id limit 1;
END
```

Testing Tool result:

```
Command Issued: hcldbprotocoltest -e ASCII -f nl simple_sp_call1 C:/cloverleaf/cis6.2/integrator/mysql/data/data.dat
```

Command output:

```
outbound db thread
Test with MESSAGE 1
1,John,Smith,2014
The output value of <GetPatientById.fname> is: John
The output value of <GetPatientById.lname> is: Smith
The output value of <GetPatientById.dob> is: 250173
Running DB Write action succeeded.
```

SQL result:

```
{call GetPatientById(1, fname OUT, lname OUT, dob OUT)}
```

SQL configuration

You can define SQL statements that are run upon a successful database connection.

The **Inbound (Outbound) SQL Statement Configurator** dialog box supports SQL statement customization.

Note: The Database protocol supports the SQLite database, which is the default.

Configuring a condition

To compose condition clauses, click **Configure** on the Options panel of the **Inbound (Outbound) SQL Statement Configurator** dialog box.

The **Inbound (Outbound) SQL Statement Configurator** dialog box can access all tables and views in the current database connection. All tables in the database are supported for select, insert, delete, and update statements.

Note: The table schema you require might not be in the **Inbound (Outbound) SQL Statement Configurator** table schema list. If this happens, then you can insert/delete/update table schema from any connection and any table from the connection to meet your requirements.

The Table/View panel displays all the column names in the selected table/view. This is defined in the **Inbound (Outbound) SQL Statement Configurator** dialog box. Double-clicking an item appends the item name to the cursor location in the Condition panel.

The Outbound Staging Message Fields panel displays all field addresses in the currently selected outbound staging message format. This format is defined in the **Database Outbound Protocol Properties** dialog box.

Double-clicking an item appends the item name to the cursor location in the Condition panel. The name is enclosed in angle brackets.

The Column/Value panel displays the available columns and values for the selected table/view:

- The Column displays the full column name in the format [table/view_name].[column_name].
- Value can be edited. The default value is the corresponding value of Column in angle brackets. For example, <table1.column1>.

When you select an item from the Table/View Schema list, all related column items are automatically appended in the table.

Double-clicking a table row places the column's full name in the SQL text area.

The configured condition is displayed in the **Condition** field.

Inbound protocol read success/read failure action

If the type of read success action is SQL Statement, then clicking **Configure** on the **Read Success Action** tab opens the **Inbound (Outbound) SQL Statement Configurator** dialog box. Based on the selected table schemas, you can configure the insert/update/delete SQL statement for the **Read Success Action** tab.

The behaviors of **Select** and **Configure** are the same as the ones on the **Inbound (Outbound) SQL Statement Configurator** dialog box for inbound protocol read action.

- If **Insert** is selected, then the **ColumnValue** fields are available.
- If **Update** is selected, then **Value** and **Condition** are available.
Clicking **Select** beside **Value** opens the **Column List** dialog box.
- If **Delete** is selected, then **Condition** is available.
- Clicking **OK** composes the SQL statement according to the current setting.
- If the type is **Insert**, then a standard insert SQL is composed:

```
insert into Schema Name (text_Column) values(text_Value)
```

- If the type is **Update**, then a standard update SQL is composed:

```
update Schema Name set text_Value where text_Condition
```

- If the type is **Delete**, then a standard delete SQL is composed:

```
delete from Schema Name where text_Condition
```

- If the type is **Select**, then this indicates the inbound reply message format in the Database Outbound protocol.

For the update /insert/delete SQL, you can define dynamic or static values in the Values/Condition clause.

Configure on the **Read Failure Action** tab has the same behavior as that of **Configure** on the **Read Success Action** tab.

Composing the standard query SQL statement

Clicking **OK** on the **Inbound (Outbound) SQL Statement Configurator** dialog box composes a standard query SQL statement according to the current setting:

```
select text_Column from Schema Name where text_Condition
```

SQL statement example 1

This is an example of scanning data in the Database Inbound protocol.

In this example, the Database Inbound protocol thread selects all columns from the `Table_sanity` table. Then, it updates the flag column. Updating the flag column is used to prevent reading duplicate data in the next scan.

```
Read Action: select * from Table_sanity where Table_sanity.flag=0 order by Table_sanity.ID
```

```
Read Success Action: update Table_sanity set flag=1 where ID=Table_sanity.ID and flag=0
```

Running this example in Testing Tool results in:

```
Command Issued: hcldbprotocoltest -r read
Command output:
Executed SQL:
select * from Table_sanity where Table_sanity.flag=0 order by Table_sanity.ID
Read 1 message(s) from DB.
key01,test1,1,10/01/12,0
Doing IB Read Success Action...
Executed SQL:
update Table_sanity set flag=1 where ID=key01 and flag=0
inbound db thread
```

SQL statement example 2

This is an example of inserting data in the Database outbound protocol.

In this example, the Database Outbound protocol thread inserts a record into the `Table_sanity` table. The placeholder `table.columnName` represents the column value that is parsed from the outbound Cloverleaf message.

```
Action: insert into Table_sanity( ID, strFLD, intFLD, dateFLD, flag) values(Table_sanity.ID, Table_sanity.strFLD, Table_sanity.intFLD, Table_sanity.dateFLD, Table_sanity.flag )
```

Running this example in Testing Tool results in:

```
Command Issued: hcldbprotocoltest -e ASCII -r -f nl write C:/cloverleaf/cisversion/integrator/t-dbp.sanity/test/test.txt
Command output:
```

```

outbound db thread
Test with MESSAGE 1
key01,test1,1,10/01/12,0
Running DB Write action succeeded.
Executed SQL:
insert into Table_sanity( ID, strFLD, intFLD, dateFLD, flag ) values( key01, test1, 1, 10/01/12,
0 )
Returned results:
> update count: 1

```

SQL statement example 3

This is an example of setting dynamic functions in the SQL pane of the Database Inbound or Database Outbound database protocol.

For example, a SQL statement must be run that selects * from patients where dateOfBirth is today(). This example shows how to handle today(), without hardcoding the current date in the SQL statement.

For this, each database provides its own date/time functions.

On an Oracle database, you can use sysdate or to_char(sysdate, 'format')/to_date(sysdate, 'format') to do it.

Command issued:

```
hcidbprotocoltest oracle_select
```

The SQL that was run is:

```
select * from Table_sanity where DATEFLD = TRUNC(SYSDATE,'DD')
```

Command output:

```

Read 1 message(s) from DB.
key01,test1,1,2014-10-01,1
inbound db thread

```

SQL statement example 4

In a simple_insert_with_parameters thread, a file is read with this structure:

```
John, Smith, 2014
```

The record is not added to the database. This is what displays in the test console:

```
CALL: INSERT INTO patients(patients.fname, patients.lname, patients.DOB) VALUES
(patients.fname, patients.lname, patients.DOB)
```

Testing Tool result:

```

Test with MESSAGE 1
John,Smith,2014
Running DB Write action succeeded.

```

The SQL run result:

```
INSERT INTO patients(patients.fname, patients.lname, patients.DOB) VALUES (Smith, 2014, )
```

In this case, the first value, `patients.fname`, is missing and there is a comma at the end of the statement.

In this example, the input message does not match the database schema format defined in the `simple_insert_with_parameters` thread.

The input message is:

```
John, Smith, 2014
```

The databases schema format (`patients`) is:

```
id, fname, lname, DOB
```

Because there is no DOB field given in the message, at runtime:

- `patients.fname` is replaced with "Smith".
- `patients.lname` is replaced with "2014".
- `patients.DOB` is replaced with "".

You can input a full message such as "idValue, John, Smith, 2014."

You can also specify a message with an empty "id." For example, ", John, Smith, 2014."

DICOM protocol

DICOM (Digital Imaging and Communications in Medicine), is used as the standard communication mechanism when integrating various medical products in a hospital environment. The medical products that are involved include modalities such as CT, MR, X-Ray, and others. It also includes workstations, archives, printers, and HIS/RIS devices.

The intention of DICOM is to define the communication capabilities of each product type. With this capability, products that are supplied by several vendors can be connected together. This forms an open, integrated diagnostic/treatment capability.

Encryption

The DICOM protocol can be encrypted over TLS in Windows and Linux installations. See [DICOM encryption](#) on page 626 .

DICOM SCP

DICOM SCP (Service Class Provider) receives a request from outside SCUs (Service Class Users). It then converts the request to an engine internal message, which is a raw DICOM message, and then sends the translation

thread. It also receives the response message from the translation thread, then sends the response back to the outside SCUs.

When this tab is configured, the **SCU Options** tab should be left blank.

This table shows the available options on the **DICOM Protocol Properties** dialog box's **SCP Options** tab.

Option	Description
Local AE Title	The AE title of the Cloverleaf CLSCP thread. This is the same as that configured in the SCU modality that sends messages to this thread.
Local Port	The TCP/IP port on which this thread listens. This is the same as that configured in the SCU modality that sends messages to this thread. The default is 11112.
Timeout	The time-out for DIMSE (DICOM Message Service Element).
Presentation Context	This defines which Abstract Syntax and Transfer Syntax are accepted by this thread. This depends on the type of messages that the SCU modality sends.
Allow Multiple Associations Max Association Number	When Allow Multiple Associations is cleared, Max Association Number is disabled. When Allow Multiple Associations is selected, Max Association Number is enabled. Note: This feature is supported only on Windows and Linux systems. On AIX, these options are disabled.
TLS/Configure	Select TLS to enable Configure . Click Configure to open the DICOM TLS Options dialog box. For details, see DICOM TLS Options dialog box on page 624.

DICOM SCU

DICOM SCU (Service Class User) receives the engine internal message from the translation thread. It then converts it into DIMSE and Dataset, and sends it to the outside SCP (Service Class Provider). It gets the responses from outside SCP. For each response, it converts DIMSE and Dataset to internal messages. It then writes the useful information to Driver Control, and sends these messages to the translation thread.

When this tab is configured, the **SCP Options** tab should be left blank.

This table shows the available options on the **DICOM Protocol Properties** dialog box's **SCU Options** tab.

Option	Description
Local AE Title	The AE title of the Cloverleaf CLSCU thread. This is the same as that configured in the SCP modality that receives messages from this thread.
Remote AE Title (Reply AET)	The AE Title of the SCP modality. Sometimes, there might be no Outbound Reply TPS that is defined on the CL_SCP side. In this case, any message whose Called AET, coming from association, does not equal to Reply AET is dropped by the engine.
Remote Host	The host name or TCP/IP address of the SCP modality.
Remote Port	The TCP/IP port of the SCP modality. Click Select to open the Select Port dialog box .
Timeout	The time-out for DIMSE (DICOM Message Service Element).
Presentation Context	This defines which abstract syntax and transfer syntax are accepted by this thread from the SCP modality. This depends on the type of messages that the SCP modality sends.
TLS/Configure	Select TLS to enable Configure . Click Configure to open the DICOM TLS Options dialog box . For details, see DICOM TLS Options dialog box on page 624.

DICOM TLS Options dialog box

Note: The **DICOM TLS Options** dialog box is similar to the TCP/IP protocol's **SSL** dialog box. See [TCP/IP protocol](#) on page 720.

A TLS configuration panel is available where you can configure the related fields.

The SCP and SCU options support TLS configuration on the **DICOM Protocol Properties** dialog box.

If **Require Peer Certificate** is selected in Peer Authentication, then these are required:

- **CA Path**
- **CA File**
- **Certificate File**
- **Private Key**
- **Password**

Peer Authentication

The peer certificate is checked when **Require Peer Certificate** is selected.

When **Verify Peer Certificate** is selected, the peer certificate is verified. Verification fails when no certificate is present.

When **Ignore Peer Certificate** is selected, the peer certificate is not verified.

Security Profile

Security profile cipher suites:

Security Profile	Function	Cipher suites
BCP 195 TLS	Enables TLS1.3	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 TLS_RSA_WITH_AES_128_CBC_SHA TLS_RSA_WITH_3DES_EDE_CBC_SHA
Non-downgrading BCP 195 TLS	Enables TLS1.3	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
Extended BCP 195 TLS	Disables TLS1.3	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 TLS_RSA_WITH_NULL_SHA
Basic TLS Secure Transport Connection	Disables TLS1.3	Cipher suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA
AES TLS Secure Transport Connection	Disables TLS1.3	TLS_RSA_WITH_AES_128_CBC_SHA TLS_RSA_WITH_3DES_EDE_CBC_SHA
Authenticated unencrypted communication	Disables TLS1.3	Cipher suite: TLS_RSA_WITH_NULL_SHA If TLS1.3 is disabled and the DCMTK library compiling with OPENSSL version is OpenSSL 1.1.1 or newer, then the supported TLS protocol version is TLS1.2. In CIS 20.1.2, the DCMTK version is <code>dcmtk-3.6.6</code> and compiled with <code>openssl-1.1.1k</code> .

Security Profile	Function	Cipher suites
Non-downgrading BCP 195 TLS	Disables TLS1.0 and TLS1.1	
Extended BCP 195 TLS		

DICOM encryption

DICOM encryption is configured using the **TLS** check box and **Configure** button on the **SCP Options** and **SCU Options** tabs.

Configure is enabled when **TLS** is selected. Click **Configure** to open the **DICOM TLS Options** dialog box.

For additional information, see [DICOM TLS Options dialog box](#) on page 624

Note: This feature is available on Windows and Linux.

Encryption terms

Terms used in configuring encryption:

Name	Description
CA Path	Certificate Authority directory path. This points to a directory containing CA certificate files in PEM/ANS1 format.
CA File	Certificate used for authentication. This points to a file containing CA certificates in PEM/ANS1 format.
Certificate File	Certificate file name. This is used for authentication and encryption.
Private Key	Private key file name. This is used to create digital signatures.
Password	This is used when your private key is password-encrypted. The driver uses this password to decrypt the private key before using it.

Supported ciphers

cipher_dcmtk.txt is the intersection of the OpenSSL library supported cipher list, cmd: `openssl ciphers -v "ALL:eNULL"` and DCMtk library cipher suite list, cipher name and keySize separated with white space.

Supported ciphers:

- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 256
- TLS_DHE_DSS_WITH_AES_256_GCM_SHA384 256
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 256
- TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 256
- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 256
- TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 256
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 128
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 128
- TLS_DHE_DSS_WITH_AES_128_GCM_SHA256 128
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 128
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 256
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 256
- TLS_DHE_DSS_WITH_AES_256_CBC_SHA256 256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 128
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 128
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 128
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA256 128
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA 256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA 256
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA 256
- TLS_DHE_DSS_WITH_AES_256_CBC_SHA 256
- TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA 256
- TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA 256
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA 128
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA 128
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA 128
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA 128
- TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA 128
- TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA 128
- TLS_RSA_WITH_AES_256_GCM_SHA384 256
- TLS_RSA_WITH_AES_128_GCM_SHA256 128
- TLS_RSA_WITH_AES_256_CBC_SHA256 256
- TLS_RSA_WITH_AES_128_CBC_SHA256 128
- TLS_RSA_WITH_AES_256_CBC_SHA 256
- TLS_RSA_WITH_CAMELLIA_256_CBC_SHA 256
- TLS_RSA_WITH_AES_128_CBC_SHA 128
- TLS_RSA_WITH_CAMELLIA_128_CBC_SHA 128
- TLS_RSA_WITH_NULL_SHA 0

Unsupported cipher suites

Some security profiles do not specify cipher suites. For example:

- Extended BCP 195 TLS: Additional cipher suites not permitted.
- BCP 195 TLS:
 - Non-downgrading BCP 195 TLS.
 - Basic TLS Secure Transport Connection.
 - AES TLS Secure Transport Connection: Unencrypted cipher suite (keySize=0, NULL-SHA).

LDL message xlate file

You can configure the `xlate` file to modify a DICOM data element value.

BULKCOPY is required to copy the IR tree from source to destination. After this, use COPY to modify the data element value.

Example:

```
prologue
xlt_infile: ldl 2014 test C_MOVE
who: hli1
date: July 17, 2014 11:57:32 AM CST
xlt_outfile: ldl 2014 test C_MOVE
type: xlt
version: 7.0
end_prologue
{ { OP BULKCOPY }
  { ERR 0 }
}
{ { OP COPY }
  { ERR 0 }
  { IN =WRIX }
  { OUT 1(0).C_MOVE_DATASET(0).2#0010,0010(0) }
}
```

TRXID determination

The "TRXID Determination Format DICOM" is composed of Abstract Syntax and Command Code.

For example 1.2.840.10008.5.1.4.1.2.1.2_0020, Patient Root Query/Retrieve Information Mod
el_MOVE_C_MOVE_RQ.

If the TRXID Determination is DICOM, then with a wildcard you can only use AbstractSyntax as `trxid` in the data route definition.

Example: TRXID Determination Format UPOC, TrxID used from PDU (AE Caller, AE Called, Presentation Context ID, Abstract Syntax ID), or from SOP (AE Caller, AE Called, Service Class UID, Command Code UID).

Example code:

```
proc trxid_CalledAET { mh {args {}} } {
  global HciConnName
  set module "trxid_CalledAET/$HciConnName"
  set msgSrcThd [msgmetaget $mh SOURCECONN]
  set debug 0
  set driverctl [msgmetaget $mh DRIVERCTL]
  set trxId ""
```

```
keylget driverctl CalledAET trxId
set trxId [string range $trxId 3 end]
return $trxId
}
```

Message routing

The DICOM routing functionality consists of:

- Association Mode (Handled by Dcmthk)
 - DICOM Protocol accepts only messages that match abstract syntax in NetConfig. This is checked at association context since abstract syntax is only available at this time.
 - Association context is stored in DICOM Session.
 - Command code is set to "0". This indicates "association context" for the purposes of forming TRXID.
 - SCU sends a set of transfer syntax. Cloverleaf intersects this set from the set that is present in NetConfig, only forwarding the remaining.
 - Message is routed according to TrxID (= Abstract Syntax + Command Code).
 - One presentation context is accepted by SCP. This presentation ID is saved in the DICOM session and is used to look up "Abstract Syntax during Data Exchange Context".
- Data Exchange Mode
 - Each PDU contains the presentation context ID. This is used to look up Abstract Context.
 - SCU can send multiple PDUs for a particular message. Cloverleaf collects all of them at the inbound and creates a SOP (=DIMSE + IOD). The remaining operations start only when all the PDUs for that message have arrived at the protocol and SOP is successfully created.
 - Cloverleaf looks up Abstract Context from the DICOM Session by the presentation context ID that is in the PDUs. It also looks up the command code from the DIMSE. Abstract Context + Command code = TrxID.
 - Message is routed according to TrxID.
 - At the outbound, SOP is broken into individual PDUs and sent out, keeping the SCP's PDU size limits under consideration.
- Message Reply:
 - In Phase 1, only replies from Called AET are passed through. All other replies are dropped that are not coming from Called AET.
 - Static Route is used as a route reply.
 - UPOC is used to create Reply based on Request.

Tcl example

```
proc ack_dicom { args } {
  global HciConnName ;# Name of thread
  keylget args MODE mode ;# Fetch mode
  set ctx "" ; keylget args CONTEXT ctx ;# Fetch tps caller context
  set uargs {} ; keylget args ARGS uargs ;# Fetch user-supplied args
  set debug 0 ; ;# Fetch user argument DEBUG and
  catch {keylget uargs DEBUG debug} ;# assume uargs is a keyed list
  set module "ack_dicom/$HciConnName/$ctx" ;# Use this before every echo/puts,
  ;# it describes where the text came from
  set dispList {} ;# Nothing to return
  switch -exact -- $mode {
    start {
      ;# Perform special init functions
    }
  }
}
```

```

# N.B.: there may or may not be a MSGID key in args
if { $debug } {
  puts stdout "$module: Starting in debug mode..."
}
}
run {
  # 'run' mode always has a MSGID; fetch and process it
  keylget args MSGID mh
  msgmetaset $mh DRIVERCTL "{TransferSyntax 1.2.840.10008.1.2.4.91}"
  set gh [grmcreate -msg $mh ldl 2014 test C_STORE]
  set gh_reply [grmcreate ldl 2014 test C_STORE_RSP]
  # Get fields
  set field0 [grmfetch $gh 0(0).C_STORE_RQ_DIMSE(0).0000,0000(0)]
  set field2 [grmfetch $gh 0(0).C_STORE_RQ_DIMSE(0).0000,0002(0)]
  set field120 [grmfetch $gh 0(0).C_STORE_RQ_DIMSE(0).0000,0110(0)]
  set field1000 [grmfetch $gh 0(0).C_STORE_RQ_DIMSE(0).0000,1000(0)]
  # Get DATA
  set COMMANDGROUP [datget $field0 VALUE]
  set SOPCLASSUID [datget $field2 VALUE]
  set MESSAGEID [datget $field120 VALUE]
  set AFSOPINSTANCEUID [datget $field1000 VALUE]
  puts $COMMANDGROUP
  puts $SOPCLASSUID
  puts $MESSAGEID
  puts $AFSOPINSTANCEUID
  # Store DATA
  grmstore $gh_reply 0(0).C_STORE_RSP_DIMSE(0).0000,0000(0) c $COMMANDGROUP
  grmstore $gh_reply 0(0).C_STORE_RSP_DIMSE(0).0000,0002(0) c $SOPCLASSUID
  grmstore $gh_reply 0(0).C_STORE_RSP_DIMSE(0).0000,0100(0) c "32789"
  grmstore $gh_reply 0(0).C_STORE_RSP_DIMSE(0).0000,0120(0) c $MESSAGEID
  grmstore $gh_reply 0(0).C_STORE_RSP_DIMSE(0).0000,0800(0) c "257"
  grmstore $gh_reply 0(0).C_STORE_RSP_DIMSE(0).0000,0900(0) c "0"
  grmstore $gh_reply 0(0).C_STORE_RSP_DIMSE(0).0000,1000(0) c $AFSOPINSTANCEUID
  set msg_new [grmencode -meta {DRIVERCTL {{TransferSyntax 1.2.840.10008.1.2}}} $gh_reply]
  grmdestroy $gh
  grmdestroy $gh_reply
  lappend dispList "SEND $msg_new" "CONTINUE $mh"
}
time {
  # Timer-based processing
  # N.B.: there may or may not be a MSGID key in args
}
shutdown {
  # Doing some clean-up work
}
default {
  error "Unknown mode '$mode' in $module"
}
}
return $dispList
}

```

Java UPoC example

```

package hc.cloverleaf.dicom;
import hc.cloverleaf.Logger;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;

import org.apache.commons.codec.binary.Hex;
import org.dcm4che2.data.DicomObject;
import org.dcm4che2.io.DicomInputStream;
import org.dcm4che2.io.DicomOutputStream;
import org.dcm4che2.net.CommandUtils;

import com.quovadx.cloverleaf.upoc.CloverEnv;
import com.quovadx.cloverleaf.upoc.CloverleafException;

```

```

import com.quovadx.cloverleaf.upoc.DispositionList;
import com.quovadx.cloverleaf.upoc.Message;
import com.quovadx.cloverleaf.upoc.PropertyTree;
public class HC_dicomACK extends com.quovadx.cloverleaf.upoc.TPS {
    private static int loglevel = 3;
    private static final Logger logger = new Logger ();
    String siteDir = "";
    String rootDir = "";

    /**
     * Constructor using <code>Upoc.extractAndValidateUserArguments</code> to
     * populate private member fields generated from User Code Component definition.
     */
    public HC_dicomACK (CloverEnv cloverEnv, PropertyTree xArgs)throws CloverleafException
    {
        super(cloverEnv,xArgs);
        extractAndValidateUserArguments(cloverEnv, xArgs);
        siteDir = cloverEnv.getSiteDirName();
        rootDir = cloverEnv.getRootName();
        logger.setLogLevel(3);
    }
    /**
     * The <code>process</code> method is called by the Cloverleaf engine at a
     * defined point of control, to allow user creation of messages and/or control
     * and modification of the content and the flow of a message passing through
     * the engine.
     * @param cloverEnv an opaque reference to the Cloverleaf run-time environment
     * @param context identifies which TPS-style Point of Control is involved
     * @param mode start | run | time (time is only used in protocol read TPS's)
     * @param msg reference to message being processed. This will be
     * <code>null</code> for start or time mode, unless there is
     * more than one TPS on the stack and an earlier one has
     * produced a message.
     * @return DispositionList - instructs the engine what to do with any message
     * passed into the call, and any message(s) generated within the call.
     */
    public DispositionList process (
        CloverEnv cloverEnv,
        String context,
        String mode,
        com.quovadx.cloverleaf.upoc.Message msg)
    throws CloverleafException {

        // Initialize our return value
        DispositionList dispList = new DispositionList();
        logger.setLogLevel(3);

        InputStream is = null;
        DicomInputStream din = null;
        String transferSyntax = "";
        DicomObject dob = null;

        Message ack = null;

        if (msg != null) {
            logger.log("Processing message...",3);
            try {
                is = new ByteArrayInputStream(msg.getBytes());
                //is = new ByteArrayInputStream(msg.getContent().getBytes("ASCII"));
            } catch (CloverleafException e2) {
                e2.printStackTrace();
            }
            logger.log("Message is:\n" + Hex.encodeHexString(msg.getBytes()), 3);
            try {
                din = new DicomInputStream(is,msg.metadata.getDriverctl().getString("TransferSyntax"));
                try {
                    if (din.getDicomObject()==null) {logger.log("no DicomObject", 3);}
                    dob = din.readDicomObject();
                    logger.log("DATA MESSAGE DICOM Object = \n"+dob.toString(),3);
                } catch (Exception e1) {
                    logger.log("failed",3);e1.printStackTrace();
                }
            }
            catch (Exception e) {e.printStackTrace();}

```

```

    finally {
        try {
            is.close();
        }
        catch (IOException ignore) {
        }
    }
    DicomObject response = CommandUtils.mkRSP(dob, CommandUtils.SUCCESS);
    //logger.log("RESPONSE DICOM Object = \n"+response.toString(),3);
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    DicomOutputStream dos = new DicomOutputStream(baos);
    try {
        dos.writeDicomObject(response, din.getTransferSyntax());
        ack = msg.copy();
        ack.setBytes(baos.toByteArray());
        //String meta = "{CallingAET "+sourceAETitle+"} {CalledAET CLOVERLEAF} {AbstractSyntax
        "+sopClassUID+"} {TransferSyntax "+transferSyntax+"}";
        logger.log("Response is:\n" + Hex.encodeHexString(ack.getBytes()), 3);

        is = new ByteArrayInputStream(ack.getBytes());
        din = new DicomInputStream(is,ack.metadata.getDriverctl().getString("TransferSyntax"));
        dob = din.readDicomObject();
        logger.log("REPLY MESSAGE DICOM Object = \n"+dob.toString(),3);
    }
    catch (IOException e) {
        e.printStackTrace();
        return dispList;
    }
    finally {
        try {
            dos.close();
        }
        catch (IOException ignore) {
        }
    }
    //Return our disposition list
    dispList.add(DispositionList.OVER, ack);
    dispList.add(DispositionList.CONTINUE, msg);
    logger.log("Done. Continuing message...",3);
}
return dispList;
}
}

```

Common Options tab

For **DICOM Engine Log Configuration**, click in the field to open a menu of options in ascending order:

- debug
- disabled
- error
- fatal
- info
- trace
- warn

DICOM association and networking

DICOM accesses DIMSE services through SOP. These SOPs are wrapped within an upper level networking protocol that works using PDUs (Protocol Data Units). The format of the PDUs are:

In addition to the DICOM network data transfer vehicle, PDUs can also form sessions (DICOM associations) between the Modalities (Application Entities [AE]).

Forming a DICOM association is essentially an exchange of capabilities and preferences of peer AEs. These are remembered for the duration of the DICOM association. DICOM associations are established every time a set of DICOM data needs to be exchanged.

The items that are highlighted are:

- # Peer AE pair
This indicates the SCU (Service Class User) and SCP (Service Class Provider) AE names. This is remembered by Cloverleaf for the duration of association.
- #2 Presentation context ID (request)
This is the presentation context ID of the presentation data that is requested or accepted. Presentation context ID are generated for this purpose by SCU. Accepted IDs are remembered by Cloverleaf for the duration of the association.
- #3 Abstract syntax
This is the type of data that is being sent without specifying the actual format. This is similar to SOPs in individual messages. For example, "I must send X-Ray images."
See [DICOM abstract syntax values](#).
- #4 Transfer syntax
This is the format of the data, for example, if you require to send compressed JPEGs and numbers are Big Endian. Some transfer syntax items are remembered by Cloverleaf for the duration of association:
 - Implicit VR Endian: Default Transfer Syntax for DICOM
UID: 1.2.840.10008.1.2
 - Explicit VR Little Endian
UID: 1.2.840.10008.1.2.1
 - Deflated Explicit VR Big Endian
UID: 1.2.840.10008.1.2.1.99
 - Explicit VR Big Endian
UID: 1.2.840.10008.1.2.2
See [DICOM transfer syntax values](#).
- #5 Presentation Context ID
This is the presentation context ID used to transfer data, with no mention of AE's or Transfer syntax. This ID is used by Cloverleaf to encode and decode data to additional routes.
- #6 DICOM Message
This is the actual DICOM message as indicated in "DICOM format."
See [LDL Configurator](#) on page 439.

Using the DICOM extension

The Cloverleaf DICOM extension is enabled by default.

DICOM features include:

- DICOM threads can be configured to accept only certain Abstract and Transfer Syntax combinations.
- DICOM messages can be routed based on the TrxID. The TrxID is composed of Abstract Syntax and Command Code of a DICOM message.

The thread TrxID configuration is set to DICOM in Thread Configuration.

- DICOM DIMSE headers are contained in `MessageContent`. The entire data is located in the file whose reference is placed in `UserData`.
 - DICOM DIMSE in `MessageContent` is in VRL format.
 - The DICOM protocol is available in Network Configurator.
- DICOM sends the content in the file that is referenced from `UserData` to outbound. `MessageContent` is effectively read-only.

To use DICOM:

- 1 Drag the DICOM icon to configure CLSCP and CLSCU threads.
- 2 Configure a static raw route between these threads.

Requirements

These tools are only for testing and are not required for functionality. You can use alternate tools or utilities for testing.

- ConQuest dicomlib server
<http://ingenium.home.xs4all.nl/dicom.html>
- dcmtk (c, dicom scu)
 - <http://dicom.offis.de/dcmtk>
 - <http://dicom.offis.de/dcmtk.php.en>
 - <ftp://dicom.offis.de/pub/dicom/offis/software/dcmtk/dcmtk360/bin/dcmtk-3.6.0-win32-i386.zip>
- dcm4che (java, dicom scu)
 - <http://www.dcm4che.org/>
 - <http://sourceforge.net/projects/dcm4che/files/dcm4che2/2.0.28/>
 - <http://sourceforge.net/projects/dcm4che/files/dcm4che3/3.3.2/>
- DICOM example files
<http://www.osirix-viewer.com/datasets/>
- DICOM Lib reference
<http://www.idoimaging.com/programs?order=program.percentile&func=512&readfmt=2&writfmt=2>

Sample CLSCP configuration

Press **Properties** to configure CLSCP properties on the **SCP Options** tab. Because this thread represents a CLSCP, **SCU Options** do not apply and should be left blank.

Configure these properties:

- **Local AE Title** is the AE title of the Cloverleaf CLSCP thread. The AE title must be the same as that configured in the SCU modality that sends messages to this thread.
- **Local Port** is the TCP/IP port on which this thread listen. This must be the same as that configured in the modality that sends messages to this thread.

Presentation Context defines which Abstract Syntax and Transfer Syntax are accepted by this thread. This depends on the type of messages sent by the SCU modality. The **ANY** option can also be selected.

Sample CLSCU configuration

Press **Properties** to configure CLSCU properties on the **SCU Options** tab. Because this thread represents a CLSCU, **SCP Options** do not apply and should be left blank.

Configure these properties:

- **Local AE Title** is the AE title of the Cloverleaf CLSCU thread. The AE title should be the same as that configured in the SCP modality that receives messages from this thread.
- **Remote AE Title** is the AE title of the SCP modality.
- **Remote Host** is the host name or TCPIP address of the SCP modality.
- **Remote Port** is the TCP/IP port of the SCP modality.

The Presentation Context pane defines which Abstract Syntax and Transfer Syntax are accepted by this thread from the SCP modality. This depends on the type of messages that the SCP modality sends. The **ANY** option can also be selected.

Testing DICOM

To test DICOM, set up the Conquest DICOM Server using AET DICOMTEST and port 5678.

- 1 Install the dcm4che or dcmtk library. (See the Tools section).
- 2 Download the sample DICOM images. (See "DICOM example files" in the Tools section).
- 3 Set up the site and start Cloverleaf:

```
setroot ...
hcisiteinit sitedicom
copy attached NetConfig to $HCISITEDIR/
hciengineerun -p sitedicom
```

- 4 Send a DICOM image through Cloverleaf. From external SCU, send C-Store to CL-SCP.

Using dcm4che:

```
dcm4che-3.2.0-bin\dcm4che-3.2.0\bin\storescu
-c AET@HOST:PORT FILE
```

Example:

```
dcm4che-3.2.0-bin\dcm4che-3.2.0\bin\storescu -c CLSCP@localhost:11010 IM-0001-0008.dcm
```

or

Using `dcmtool`:

```
dcmtool-3.6.0-win32-i386\bin\storescu -ll debug -xv -xw HOST PORT  
FILE
```

Example:

```
dcmtool-3.6.0-win32-i386\bin\storescu -ll debug -xv -xw localhost 11010 IM-0001-0009.dcm
```

In these examples:

- **Host** is the host/IP of Cloverleaf.
- **Port** is the port for CL_SCP. This is set to 11010.
- **AET** is the local AE Title for CLSCP.
- **File** is the `dcm` file. See the Tools section to download samples.

5 View images in the DICOM server.

DICOM example

This process shows how Cloverleaf handles a DICOM request, routes, and sends back the reply.

This example has one CL_SCP, that can receive the message to RIS, MPPS, and PACS, and routes:

- RIS message to `to_RIS`
- MPPS message to `to_MPPS`
- PACS message to `to_PACS`

The Dvtool Modality Emulator is used as an outside SCU side. All port are 11112. This is the port that is defined in the CL_SCP thread.

The AE Title that is set in the Emulator must match the Remote AE Title definition in `to_RIS`, `to_MPPS`, and `to_PACS`.

If they do not match, then the reply message is automatically dropped by the engine.

The Dvtool RIS Emulator is used as RIS (port 107, defined as Remote Port in `to_RIS`). It is also used as MPPS (port 108, defined as Remote Port in `to_MPPS`) systems.

The ConQuest DICOM server is used as a PACS system. The port is 5678, defined as Remote Port in `to_PACS`. `StorageCommitmentPushModelSOPClass` is not supported, so you must use another server on which to test Store Commit.

The **TRXID Determination** is DICOM. Using a wildcard, you can only use the AbstractSyntax as `trxid` in the data route definition, and `static` as the reply route.

Note: The example messages come from the default DVTK.

To test:

- 1 Configure and start the Modality Emulator, RIS Emulator, and ConQuest DICOM Server.
- 2 Start process "Vivian".
- 3 Click **Request Worklist** in Modality Emulator. This returns the results.

- 4 Select one of the records.
- 5 Store Image in Modality Emulator. When it is sending, you can open the ConQuest log. ConQuest receives several stored messages.
- 6 Send **MPPS Completed**.

DTC

Distributed Transaction Controller (DTC) provides overall transaction orchestration, where users can implement a state machine in Cloverleaf.

With DTC, messages can be temporarily stored, translated, enriched by querying external systems or otherwise and sent when ready for outbound.

Using DTC, users have control to commit, rollback, or proceed to the next step depending on success or error in the transaction states. The transaction states and associated actions are user defined.

DTC transactions are tracked in a site-specific database, which gives DTC the capability to work across all processes and threads in a site.

These major components are part of a transaction system:

- Binary engine:
This runs the list of steps in a pre-determined order. There are various protocols that the binary engine follows, such as 2-phase commit, 3-phase commit and Paxos.
- List of steps to run:
This is a pre-determined ordered list. For example, by a query planner, and handed to the binary engine. Each step runs successfully or with an error. If an error, then the binary engine rolls back the transaction by undoing the result of the previous successful steps.
- Rollback journal:
This stores undo information, which is stored on disk. This is a special table that is accessible only by the binary engine, and stores the values before a step changes it. On running the command, the binary engine first copies the step's *main data* into the rollback journal and then runs the step. If the step was successful, then it moves to the next step. If unsuccessful, then the binary engine overwrites the *main data* with the value from the rollback journal for this and all previously run steps.

DTC provides coordination and transaction functionality between several independent threads that may be in the same Cloverleaf instance.

With DTC, messages can be temporarily stored, enriched by querying external systems or otherwise, and sent when ready for outbound.

DTC auto-commits. Users have control of proceeding to the next step or rollback, depending on success or error in the transaction states. The transaction states and associated actions are user-defined.

DTC is implemented as a protocol in Cloverleaf. DTC threads are the binary engine that runs the transaction.

On the DTC's IDE, users can create the list of steps in the form of a state machine. Action TPS are run with each step and govern the flow of the state machine, providing an order for the list of steps. In other words,

users are the query planner when they use the DTC's IDE to build the state machine and write Action TPS. Rollback TPS cannot be used to handle error conditions.

A Staging DB is created to handle the Rollback Journal functionality. This is a general feature and can be used without DTC. The Staging DB stores key value pairs under an umbrella key space. Each transaction creates its own key space and support for global key space is available.

Engine commands are provided to assist with debugging. For example, when `*dtc_debug step*` is set, the binary engine waits before running the next step. It only does so when `*dtc_debug step*` is run from the command line.

CLAPI support is included to configure the DTC thread and properties.

Information storage

Distributed Transaction Controller files are stored in `$HCISITEDIR/dtc`.

The staging database, `$HCISITEDIR/exec/databases`, is used for DTC state tracking.

DTCCTX

DTCCTX is a TCL keyed list that contains information about the current state. DTCCTX is copied into the DTC thread's DRVCTRL before calling the Action and Rollback TPS.

TPS's can then extract DTCCTX from DRVCTRL to read and write, to or from, the Staging database.

The DTC specific Staging database TCL API accepts DTCCTX as a parameter.

DTCCTX has this structure:

```
{DTCCTX { {XID xid} {STEP step} {STATE state} } }
```

Key	Description
XID	Transaction ID
STEP	Current step number
STATE	State name

Testing Tool usage

The Testing Tool gives visibility into the Distributed Transaction Controller so that the interface developer can see transactions in each state of the data orchestration. The route test has been enhanced to enable this detailed view.

If required, then you can set break points in the testing data flow, where transaction details can be viewed at each breakpoint.

DTC Protocol Properties dialog box

To configure DTC, drag the DTC icon to the layout pane. Click Protocol **Properties** to open the **DTC Protocol Properties** dialog box.

These tabs are available for configuring DTC protocol properties:

- General
- States
- Transitions

General tab

This tab contains these parameters:

- **Enable Transaction**

Select this to enable/disable rollback procs if an error happens in the DTC thread. By default, this is selected.

When a message is received at Stop State, DTC initiates these operations:

- The message is in an error state:
 - 1 If transactions are enabled, then the Rollback TPS is called in reverse order on each state as entered in this transaction.
 - 2 Transaction information is deleted from the DTC.
 - 3 The message is then sent to the error database.
- The message is not in an error state:
 - 1 The message is passed to the Action TPS with the transaction ID and DTC Mode “Data”.
 - 2 The Action TPS is run. This returns a Transition Code.
 - 3 If the Transition Code is an error, then the error state steps are followed.
Otherwise, the transaction info is deleted.

- **Max Steps**

The initial value is 1000. This must be defined. Choose a number that is large enough for your logic to run. The goal is to limit the possibility of an infinite loop in DTC processing.

- **XID Proc**

The DTC protocol automatically generates the XID. When it is the scope of the DTC, the XID is used as the unique identifier for the message. You can override the default XID that is generated by implementing an XID proc. This is the same as a txid proc. You should let the DTC assign the XID.

- **Timeout**

This configuration is used to check the messages in the staging database. If a message stays in the staging database longer than the time-out value, then DTC rolls it back. The default value is 30. The DTC timeout value is limited to 8-bit numbers (0-99999999).

In previous versions, DTC lacked a time-out mechanism, so that a transaction could stop responding if the waited reply never came. The DTC protocol thread cannot deliver messages across processes. Only xlate threads can do that. For DTC threads, sending a message to another process creates a panic. The DTC timer rolls back messages that are stored in the staging database for a long time.

In some situations, certain states might occur in a loop. Because of errors, the loop could run continually. To detect this situation, the total steps in the state machine are no more than the defined max steps.

When a state is entered, it is saved in the staging database. Current count of steps can be found with this query:

```
select max(step) from stage where datatype = 'DTC' and keyspace = XID
```

You cannot change or alter the steps in the DTC table.

States tab

This tab contains a table listing all configured states for the DTC protocol thread.

By default, DTC has two initial states: START and STOP. These are pre-filled on the dialog box. These states are required. They can be edited, but cannot be deleted by users. An icon before the state name indicates the state type: START, STOP, or user-defined.

Define the required states/stops for your message processing -- not the order, but only the steps.

A TPS generates transaction IDs. If left blank, then transaction IDs are auto-generated. This is independent of whether transactions are enabled.

Actions are optional, in which case the message is routed directly to the thread.

This tab contains these editing options:

- **Add** opens the **Add DTC State** dialog box. This has text fields for **State Name**, **Action TPS**, and **Rollback TPS**.
The TPS fields also have an **Edit** button that opens the **TPS Editor** dialog box for updating the TPS field.
- **Edit** is enabled when a state is selected in the state table. Clicking the button opens the **Edit DTC State** dialog box. This is the same as the **Add DTC State** dialog box, except it is populated with the selected DTC state.
- **Delete** is enabled when a state is selected in the state table. Clicking the button removes the state from the table, and the next state or last state becomes selected.

This table shows how you can model individual states within a transaction by entering sample information:

State	Action	Next State	Rollback
1	Action1.tcl	Action 1 == State 3 Action 2 == State 4 Else – State 2	Rollback1.tcl
2	Send to thread X	After thread X responds – State 3	Rollback2.tcl
3	Action3.tcl	Go to State 4	Rollback3.tcl
4	Action4.tcl	Go to State 5	Rollback4.tcl
5	Action5.tcl (optional)	Commit	Rollback5.tcl

The Action column defines the work that is performed when in that state.

For the Next State column, the state transitions, which are modeled in the user interface, are governed by the return value of the Action step. Message translation is available during the Send to thread action. Receiving threads may belong to a process other than DTC.

The Rollback column defines the work to do when the Distributed Transaction Controller sends a rollback message. Each state does its own cleanup, for example, when Action created files locally on disk. Distributed Transaction Controller is responsible for cleaning up the staging database and state information known by Distributed Transaction Controller.

Transitions tab

This tab contains a table listing all configured transitions for the DTC protocol thread.

This tab defines the flow of the message through the DTC. This is based on If/Then statements.

The values in the Code column are user-defined and must be in sync with the Tcl procs that are defined in the **States** tab.

Clicking **Add** opens the **Add DTC Transition** dialog box. When **Transite to State** is selected, the **Code** and **State** text fields are enabled.

- For state transitions, you can define any alpha-numeric codes, as long as it does not conflict with HTTP codes.
- When **Send to Thread** is selected, the **Thread** menu and **Reply Trxid** field are enabled.
- When the DTC protocol thread is saved, a reply route is created for all “send to thread” transitions.
- If **Reply Trxid** is empty, then a static route is created; otherwise, a route with the specified trxid is created.
- **Reply Trxid** is used to create a reply route from the outbound thread to this DTC thread. The DTC thread knows if a message is sent out successfully by the outbound thread. Then, it goes to the next action.
- You can add routes to other threads from the DTC thread. These thread names can be selected in response to “SEND” Transition Codes from the Properties UI.
- In some situations, a user’s use case might only do processing and not route messages in a particular state.

Edit is enabled when a transition is selected in the transition table. Clicking this button opens the Edit DTC Transition dialog box. This is the same as the **Add DTC Transition** dialog box, except it is populated with the selected DTC transition.

Delete is enabled when a transition is selected in the transition table. Clicking this button removes the selected transition from the table, and the next transition or last transition becomes selected.

Data flow

A NetConfig with DTC has these threads:

- DTC is the main engine that runs the ordered list of steps in the form of a State Machine. This is defined by the user through the DTC IDE GUI.

- "Inbound" is any thread that sends DATA to a DTC thread. When DATA is received at a DTC thread, a new transaction is created. There can be multiple threads sending DATA to a DTC thread and each instance of DATA creates a new transaction.
- "Intermediate" are those that receive messages from DTC threads depending on the state transitions. Some states may send a message to their corresponding thread. Some transition to the next state without sending a message. This behavior is programmable through the Action TPS.
- "Outbound" is the last thread that receives the message. A transaction ends when the outbound thread sends a successful reply message to the DTC thread. The DTC thread then enters a stop state. The outbound thread and successive transition to a stop state is configured through the DTC IDE GUI.

Data flow modes

This table lists the modes of data flow:

Mode	Description
Inbound thread→DTC thread	A new Transaction (XID) is created and the Start state is entered with DATA. Control is passed to Start state's Action TPS. This can then return the disposition of DTCNEXT or DTCSEND.
State A→State B	State A's action returned a disposition of "DTCNEXT {\$mh Code}". The state machine had a rule of "State A + Code→State B." State B is entered and control is passed to state B's Action.
State A→Thread A	State A's Action returned a disposition of "DTCSEND \$mh" and state machine had a rule for "State A + DTCSEND→Thread A." A message is sent to Thread A.
State A→Xlate Thread→Thread A	State A's Action returned a disposition of "DTCXLATE \$mh." A message is sent to xlate thread which can then translate and route to the appropriate thread. _{{ {DTCXLATE \$mh}_}} sends the message to the DTC translation thread. A data route must be configured between the DTC thread and the destination protocol thread.
Thread A→State A (reply)	Thread A sent a reply message to DTC Thread. State machine had a rule for "State A + DTCSEND→Thread A." State A is reverse looked up from this rule and State A is entered and state A's Action TPS is called.
State A→Stop state and no rule for "Stop State + DTC-SEND→Thread X" exists	Stop state's Action TPS is called. All journal data and user data that are associated with this transaction are deleted and the transaction is deemed to be committed successfully.

Mode	Description
State A→Stop state and a rule for “Stop State + DTC-SEND→Thread X” exists	<p>Stop state’s Action TPS is called, which returns a disposition of DTCSSEND. A message is sent to Thread X. This is the outbound thread.</p> <p>The outbound thread sends a reply message to the DTC thread. The thread does a reverse lookup state by the sender thread.</p> <p>Stop state is entered with REPLY mode and Stop state’s Action TPS is called. All journal data and user data that are associated with this transaction are deleted and the transaction is deemed to be committed successfully.</p>
Error Anywhere→Stop State	<p>These types of error can occur:</p> <ul style="list-style-type: none"> Action TPS of any state returns an ERROR disposition. State + Code combination is unknown: There is no rule stating “State A + Code→State B” for Code returned by State A’s Action TPS. Max Steps limit is exceeded. <p>Stop state is entered with ERROR. The original message at Start state is sent to the Error DB and the Rollback TPS is called for each state visited in reverse order.</p> <p>All journal data for the transaction is deleted and the transaction is deemed to be rolled back.</p>

IDE configuration

To begin configuration, you can drag the DTC icon on the NetConfig or select from the **Protocol** menu. Specific properties are:

- A list of DTC states, with options to add and delete states.
 - Option to mark one state as a Start, another as a Stop state, and another as a Wait state.
 - A user interface to define a state transition.
 - A check box to indicate if transactions are enabled.
 - Add Action, Rollback TPS to each state.
- Actions are optional, in which case the message is routed directly to the thread.
- A TPS to generate transaction IDs. If left blank, then transaction IDs are auto-generated. This is independent of whether transactions are enabled.
 - Max Steps

This table shows the options for different states:

Next		State			
State	Action		Code	State/Thread	Rollback
START	StartAction.TP S	DATA	COMPLETE ENRICH	OUTBOUND EMP	StartRollback. TPS

		Next	State		
State	Action		Code	State/Thread	Rollback
EMPI	EMPIAction.TPS	DATA	COMPLETE	OUTBOUND	EMPIRollback.TPS
			ENRICH	DEMOGRAPHICS	
DEMOGRAPHICS	DemoAction.TPS	DATA	COMPLETE	OUTBOUND	DemoRollback.TPS
		REPLY	ENRICH	QUESTIONAIRE	
			SEND	demographics_thread	
			COMPLETE		
			ENRICH	OUTBOUND	
				QUESTIONAIRE	
QUESTIONAIRE	QAction.TPS	DATA	COMPLETE	OUTBOUND	QRollback.TPS
		REPLY	SEND	questionnaire_thread	
			COMPLETE		
				OUTBOUND	
OUTBOUND	OutboundAction.TPS	DATA	SEND	outbound_thread	OutboundRollback.TPS
		REPLY	COMPLETE		
				STOP	
STOP	StopAction.TPS				StopRollback.TPS

By default, DTC has two states, a Start state and an End state. These are pre-filled in the **DTC Properties Properties** dialog box.

States can be added and deleted by clicking the appropriate button.

Routes can be added to other threads from the DTC thread. These thread names can be selected in response to SEND transition codes from the Properties dialog box. You can have a use case that might only do processing, and not route messages in a particular state.

Staging database

The staging database is a global database that you can use to store keys and value pairs under key spaces. This database supports creating and deleting key spaces and keys, reading and updating values of keys.

This database is shared between threads. To avoid locking issues, a queue based producer-consumer model is built. In this model, the staging database is controlled by a thread and reads requests from a queue. Each request has a callback that is called when the request is completed. The thread is an operating system thread and not a Cloverleaf engine thread.

By default all keys are written to the global key space. DTC creates a new key space for each transaction.

Staging database access from TCL is through the stageXXX TCL API.

This table shows the key/value pairs of the staging database:

	Field	Name	Type	Notes
1	Row ID	rowid	INTEGER	SQLite default
2	Timestamp	timestamp	INTEGER	Use SQLite's <code>strftime('%Y-%m-%d %H:%M:%f', 'now')</code> function to store value. %f stores milliseconds.
3	Value type	datatype	INTEGER	<p>This field can take these values:</p> <ul style="list-style-type: none"> DTC for (key,value) DTC protocol writes: <ul style="list-style-type: none"> Data field contains Message Content Metadata field contains Message's Metadata MSG for user (key,value) when value is a message: <ul style="list-style-type: none"> Data field contains Message Content Metadata field contains Message's Metadata DAT for user (key,value). Data field contains value. Metadata field is NULL
4	Key space	keyspace	VARCHAR	<p>"Global," or user-defined key space.</p> <p>For DTC, this is "thread^XID."</p>

	Field	Name	Type	Notes
5	Key	key	VARCHAR	<p>User-defined key. For DTC, this is "step^state." These SQLite functions can be used for DTC:</p> <ul style="list-style-type: none"> <code>instr(X,Y)</code> This returns offset of string Y in X. <code>substr(X,Y,Z)</code> <code>CAST(X as TYPE)</code> This casts one type into another. For example, <code>CAST('1000' as Integer)</code>. This returns substring in X starting in Y of Z length. Z is optional. <code>nullif(X,Y)</code> If <code>nullif(X,Y)</code> returns 0 (Y was not found in X), Null is returned. <p>To get the state, use: <code>select(substring(key, 0, instr(key,"\$") + 1)</code></p> <p>To get the step: This returns substring in X starting in <code>select(CAST(substring(key, instr(key,"\$") + 1) as INTEGER)</code></p>
6	Data	data	BLOB	Message or user-defined data.
7	Meta Data	metadata	VARCHAR	Message metadata.

Queries

This table shows query types:

	Tcl proc	Description
1 DTC/user	<code>delete from stage where xid = ?</code>	Delete Keyspace. Used at Stop transaction.

	Tcl proc	Description
2 DTC	<pre>select state, step from stage where keyspace = ? order by step desc</pre>	Used at rollback transaction.
3 DTC	<pre>select ifnull(max(step), 0)+1 from stage where keyspace = ?</pre>	Get next step value when entering a state.
4 DTC/user	<p>Insert into stage (keyspace, timestamp, datatype, data, metadata, state, step) values (?,?,?,?,?,?,?)</p> <p>DTC inserts with key space=<XID>, key=<state\$step>, datatype='DTC', data=<message> metadata=<meta-data>, state=<state> and step=<step></p> <p>User inserts message with keyspace=<XID/ Keyspace>, key=<some-key>, datatype='MSG', data=<message>, metadata=<meta-data>, state=NULL and step=NULL</p> <p>User inserts value with keyspace=<XID/ Keyspace>, key=<some-key>, datatype='DAT', data=<value>, metadata=NULL, state=NULL and step=NULL</p> <p>Select data from stage where keyspace = ? and key = ? and datatype = 'DAT'</p> <p>Select data, metadata from stage where keyspace = ? and key = ? and datatype = 'MSG'</p>	<p>Add key,value to a keyspace</p> <p>Keyspace: XID, global or user defined.</p> <p>key:</p> <ul style="list-style-type: none"> state\$step for type is DTC. Alpha-numeric: user provided key. <p>timestamp: Use SQLite's strftime('%Y-%m-%d %H:%M:%f', 'now') function to store value.</p> <p>%f stores milliseconds.</p> <p>datatype:</p> <ul style="list-style-type: none"> DTC for DTC protocol values. MSG for user's message data. DAT for user data. <p>data:</p> <ul style="list-style-type: none"> Message blob for type is DTC or MSG. String value for type is DAT. <p>Get value by key from a keyspace.</p> <p>Get message by key from a keyspace.</p>

Staging database Tcl API

This table shows the (key,value) that are stored per thread:

	Tcl proc	Description
1	stageinit (processname)	Opens the staging database of the specified process. This is only required when called from <code>hcitcl</code> .
2	stageremovekeyspace (keyspace)	Removes the keyspace and all its associated data. You cannot delete global keyspace.
3	stageget (keyspace, key)	Gets a user-defined value that is stored in the staging database. Key is any alpha numeric value.
4	stageset (keyspace, key, data)	Sets the user-defined value. The previous value is overwritten.
5	stageremove (keyspace, key)	Removes the user-defined key/value pair.
6	stagegetmsg (keyspace, key)	Reads the message that is associated with the key from the staging database and returns a message handle.
7	stagesetmsg (keyspace, key, msg_handle)	Writes the message that is associated with the message handle into the staging database.
8	stagegetkeys (keyspace)	Gets all keys in the keyspace.

DTC Tcl API

This table shows the (key,value) that are stored per DTC network. These contain multiple threads within a process:

	Tcl proc	Description
1	dtcset (dtcctx, key, value)	Sets a user defined value in staging database for key.
2	dtcget (dtcctx, key)	Gets a user defined value that is stored in staging database for key.
3	dtcsetmsg (dtcctx, key, msgId)	Writes the message that is associated with the message handle into the staging database.
4	dtcgetmsg (dtcctx, key)	Reads the message that is associated with the key from staging database and returns a message handle.

Request object

The Staging database is a global database shared between threads. To avoid locking issues, a queue based producer-consumer model is built. In this model, the staging database is controlled by a thread and reads requests from a queue.

Each request has a callback that is called when the request is finished. The thread is an operating system thread and not a Cloverleaf engine thread.

This table shows the request object fields:

	Field	Name	C++ type	Notes
1	RequestType	reqtype	integer	One of: <ul style="list-style-type: none"> • CREATE_KEYSPACE • REMOVE_KEYSPACE • SET_KEY • GET_KEY • REMOVE_KEY
2	Key Space	keyspace	string	
3	Key	key	string	
4	Data	data	string	
5	Metadata	metadata	string	
6	Value Type	datatype	integer	DTC for (key,value) DTC protocol writes: <ul style="list-style-type: none"> • Data field contains Message Content • Metadata field contains Message's Metadata MSG for user (key,value) when value is a message: <ul style="list-style-type: none"> • Data field contains Message Content • Metadata field contains Message's Metadata DAT for user (key,value). Data field contains value. Metadata field is NULL.
7	Return Value	ret	integer	STAGE_ERR_SUCCESS(0) If success, other values mean failure.

DTC debugging engine commands

Debugging functionality is provided through engine commands. When debugging is on, DTC yields before entering a new state. Users can query the state's information or continue running the state.

The general syntax for the DTC debugging command is:

The general syntax for DTC debugging command is:

```
conn dtc_debug [{on [XID]|off|step|info|retry}]
```

- `conn` is the DTC thread name. This is required.
- `XID` is the transaction ID to debug. This is optional. If this is not provided, then DTC breaks for all XIDs.

Support for the XID Generator TPS is provided to assist with debugging. When you use this TPS, you can generate a custom XID. This can be used in debugging commands as a breakpoint value.

DTC debugging example

In this example, the `hl7_in` thread is configured as "not auto-start."

- 1 Start the engine. From the command line, run `hciengine -nl -p hie`.
- 2 Start `hciecmd` and run `hciecmd -p hie`.
- 3 Run `hciecmd HIE dtc_debug on`.
- 4 Run `hciecmd hl7_in pstart` to start thread `hl7_in`.
- 5 Run `hciecmd HIE dtc_debug info`.

`hciecmd` output:

```
[0.0.11634] XID=5742782c4da30 currState=START, Step=1, Disp=DTCNEXT,
Code=ENRICH, nextState=state1
```

- 6 Run the `hciecmd HIE dtc_debug step` and `hciecmd HIE dtc_debug info` several times. The state transition information is shown in the `hciengine` output and the `hciecmd` output.

Command descriptions

This table shows the command descriptions:

	Command	Description
1	<code>on [XID]</code>	Turn DTC debugging on. If XID is provided, then DTC is turned on only for that XID.
2	<code>off</code>	Turn DTC debugging off.
3	<code>step</code>	Go to the next state.
4	<code>info</code>	Print info for the current state. DTCCTX values, disposition, transaction code, and next state are printed to screen.

Use case

A user must build an interface that requires queries to multiple data sources that are based on a single input message.

The consolidated message is received by the downstream system as a single transaction. This contains data collated from the multiple data sources.

Receiver scenario

This user is an Information Developer building an interface that receives an event notification message in XML format that is received by a Cloverleaf thread.

In a secondary message (Step 7), the questionnaire (Step 5) is combined with the demographics (Step 4). This is then sent to a separate web service API end point.

The order of operation is:

- 1 The interface creates a DTC transaction upon receipt of the event notification.
- 2 The interface extracts the enterprise ID from the transaction and performs a web-service query to an enterprise main patient index (EMPI) service. This service returns all of the IDs that are known for the patient. This list of IDs includes an encounter ID.
- 3 Using DTC, the interface stores these IDs for later use in the DTC staging database, keyed on the DTC transaction ID.
- 4 The interface makes an invocation to a second web service. This service returns the patient resource (demographics), returned in XML format. This demographic information is stored in the DTC staging database, keyed on the DTC transaction ID.
- 5 The interface makes an invocation to a third web service which returns a questionnaire ID. This ID is stored in the DTC staging database.
- 6 When all web service invocations are complete, this data from the DTC staging database is collated to create a single outbound transaction:
 - Enterprise ID
 - Patient demographic resource
 - Encounter ID
 - Questionnaire ID
- 7 Secondary message delivery. The questionnaire results are combined with the demographics retrieved earlier. Cloverleaf picks up these stored values and enriches the outbound questionnaire transaction with these IDs.
- 8 The outbound message is delivered to the downstream system for processing.

Testing the interface

The user must test the interface that was built in the receiver scenario.

- The Testing Tool is used to verify all data in and out of the DTC process.
- Breakpoints are set at each stage in processing. This validates that the data stored in the staging database applies to the scenario being built.
- Users can view each outbound transaction and inbound reply used during processing.

Examples

Add the sample TPS procs for processing and rollback.

Two states are used in processing:

```
START -> state_1
state_1 -> state_2
state_2 -> STOP
```

- DTC generates the XID and forwards a new message to START.
From Start if DONE is returned.
- DTC forwards the message to state_2.
From Start if ENRICH is returned.
- DTC forwards the message to state_1.
 - state_1 forwards all messages to a thread named pix.
 - state_1 returns all replies with a code of DONE.
- DTC forwards to state_2.
state_2 returns all messages with a code of FINISH.
- DTC forwards to STOP.

DTC Tcl procs are normal TSP procs. DTC procs overload the dispList with additional properties.

Some metadata is modified by the Tcl procs.

- start.tps

```
set xCode ENRICH
lappend dispList "DTCNEXT {{$mh $xCode}}"
```

- state_1 forwards to thread pix.

```
dtcset $dtcctx mykey1 $data
lappend dispList "DTCSEND $mh"
```

- The pix thread receives the message from state_1, modifies the message, and returns the message to state_1.

```
msgmetaset $reply DESTCONN HIE
lappend dispList "OVER $reply"
```

- state_1 receives replies from thread pix.

```
set xCode DONE
lappend dispList "DTCNEXT {{$mh $xCode}}"
```

- state_2 forwards all message to last step.

```
set data [msgget $mh]
#Notify DTC that message is finished
set xCode FINISH
```

```
lappend dispList "DTCNEXT {{$mh $xCode}}"
# create a new message and send it to xlate thread
set mh2 [msgcreate $data]
lappend dispList "OVER $mh2"
```

State and Step

Characteristics of State and Step are:

- There must be at least two states of type `START` and `STOP`.
When a new message arrives, `START` state is entered and a unique `XID` is generated for the transaction.
If any error happens, then `STOP` state is entered to do rollbacks.
- Every state is configured with an Action TPS and a Rollback TPS.
- A synchronous state can perform all its work in the action TPS.
- An asynchronous state requires an associated thread's help. When the asynchronous state is entered, the DTC thread sends a `DATA` message to the associated thread. It gets necessary information from the `REPLY` message sent back by the associated thread.
- The DTC state machine runs in the outbound side `writeMsg` function. This is `pd_fn_write_msg` in the protocol driver. For example, after `OB Post-TPS Queue`.
- Step increases by one for every state transition. When a state is entered, the `DATA` message is automatically stored in the staging database and then the Action TPS is called.

Action TPS

Characteristics of the Action TPS are:

- An Action TPS can only have one `Tcl` proc.
- When a state is entered, its Action TPS is called with an input message argument. The message is a `DATA` message, and is automatically saved in the Staging Database.
- `DTCCTX` is stored in the input message's metadata `DRIVERCTL`:

```
keylget args MSGID mh
set drvctl [msgmetaget $mh DRIVERCTL]
set dtcctx [keylget drvctl DTCCTX]
```

- Use `dtcset/dtcget` and `dtcsetmsg/dtcgetmsg` to store and retrieve a value or message to or from the staging databases.
- Use `msgcreate/msgcopy` to generate new messages.
- Append a `{DISP ...}` to the return value `dispList` for every message, including input and new messages.
- `{DISP ...}` can be one of:
 - `{KILL $mh}`: Destroy the message.
 - `{OVER $mh}`: Put the message into DTC thread's IB Pre-TPS Queue. The message eventually goes to the `xlate` thread.
 - `{ERROR $mh}`: Enter `STOP` state, rollback and put the original message into the error database.
 - `{DTCSEND $mh}`: Send the message to the associated thread by ICL.

- `{DTCNEXT {{ $mh $code }}`: Return `$code`. DTC transits to the next state.
- In `dispList`, there must be one and only one `DTCSEND` or `DTCNEXT` or `ERROR`, but there can be more than one `KILL` or `OVER`.
- When an asynchronous state is entered, the Action TPS is called with a `DATA` message. The `dispList` must include a `{DTCSEND $mh}`. DTC then sends the message `$mh` to the associated thread.
- The associated thread should send back a `REPLY` message with the same `DTCCTX` in the `DRIVERCTL`.
- When the `REPLY` message arrives, the Action TPS is called the second time. In it a new `DATA` message `$mh` must be created and `{DTCNEXT {{ $mh $code }}` must be included in `dispList`.

Rollback TPS

A Rollback TPS can only have one Tcl proc.

If in a state the Action TPS returns `{ERROR $mh}`, then DTC calls every Rollback TPS for the current step and all previous steps in reverse order. The `DATA` message of that step is stored in the staging database as an input argument.

The Rollback TPS should always return `{CONTINUE $mh}`. `$mh` is the input `DATA` message.

Fileset FTP protocol

Use the Fileset FTP protocol driver for file-based interfaces. It handles groups (directories) of files, locally or remotely by FTP. It is capable of reading and writing messages contained in files, both locally and remotely, using FTP (File Transfer Protocol).

The libcurl third-party package is integrated into the system for secure implementation. This is the name of the underlying library that is used to implement cURL.

The Fileset FTP protocol has a Scheduling pane that contains the controls for scheduling: **Use advanced scheduling** and **Setup**.

The driver can process files containing multiple messages, saving state to continue at the correct point across stop/start boundaries. After all messages within the file or files are read by the engine, the source file is deleted.

- On input, supply a Tcl Procedure Stream (TPS) to parse directory listings and determine file processing order. Provide another TPS that determines which files are removed.
- On output, the driver relies on message metadata to determine the output file name.

For FTP operations, the driver assumes servers are RFC 959 (File Transfer Protocol) compliant.

Pre-configuration

Configuring this driver requires some knowledge of the remote FTP server with which it connects.

You must know:

- What user/password/account log-in information is required.
- The kind of data (ASCII or binary) to transfer.

- Whether the server supports the appropriate type.

Note: Fileset-FTP and Fileset Local populate DRIVERCTL metadata with the Inbound file name as a keyed list entry {FILENAME xxx}.

- For inbound, you must know how to generate a file list and how to parse that list so your TPS can return an ordered list. Most servers support the NLST and LIST commands.
 - The NLST command returns only the list of files.
 - The LIST command produces a long listing; the formatting and data available depend heavily on the FTP server and the operating system it is running under.

Some servers permit options to be specified at the end of the NLST/LIST command line.

- For outbound, determine whether the FTP server supports the APPE command. If it does not, then avoid using the OBAPPEND flag so that the driver generates only STOR commands.

Information is available on the configurable parameters. See:

- [Fileset/FTP Local Inbound pane](#)
- [Fileset/FTP Local TPS pane](#)
- [Fileset/FTP Local Scheduling pane](#)
- [Fileset/FTP Local Outbound pane](#)
- [Fileset/FTP Local Start-Up Procedures pane](#)

Scanning multiple directories for inbound pickup

You can scan multiple directories for inbound pickup in the fileset-local, fileset-ftp/ftps/sftp protocols and during BOX creation using these protocols.

The multiple directories are separated by white spaces.

If you require an inbound directory separated with spaces, then use curly braces around the path. For example, {/home/hci/ib dir}.

To scan multiple directories, for example, /home/hci/dira, /home/hci/dir b, and /home/hci/dirc, use this configuration: /home/hci/dira {/home/hci/dir b} /home/hci/dirc.

Fileset FTP protocol: Fileset Options tab

Inbound pane

This table shows inbound configuration parameters:

Parameters	Description
Directory	Specify the inbound file directory path. You can also specify multiple directories. See Scanning multiple directories for inbound pickup .

Parameters	Description
Style	<p data-bbox="818 300 1409 401">Click the arrow to select from the list how messages are stored within the files. This determines how the driver processes each file.</p> <ul data-bbox="818 411 1409 548" style="list-style-type: none"> • If single, then the driver reads the contents of the file into a single message. • hl7 and nl specify different schemes for delimiting messages within the file. <p data-bbox="867 562 1409 663">For nl style, the delimiter is <NL> between messages. As each message ends with <CR>, between messages would look similar to <CR><NL>.</p> <p data-bbox="867 678 1019 709">For example:</p> <pre data-bbox="883 751 1393 779">MSH <CR><NL>MSH <CR><NL>MSH <CR><NL></pre> <p data-bbox="867 814 1409 951">File read/writes in nl mode do not support the Unicode UTF-16 encoding. The nl mode cannot be performed without knowing the encoding beforehand, which is not known.</p> <p data-bbox="867 961 1409 1377">For hl7, the delimiter is <CR>MSH and the last character at end-of-file must be NL. The NL at the end-of-file indicates that it is the end of the message. Basically, each message ends with <CR>. With the hl7 style there must be <CR>MSH in-between each message. The MSH is technically not a delimiter because it belongs to the next message. It serves only as the indicator for the beginning of the next message during parsing. There are no delimiters between HL7 messages, as the engine seeks the <CR>MSH to find the beginning of the next message.</p> <p data-bbox="867 1388 1019 1419">For example:</p> <pre data-bbox="883 1461 1317 1488">MSH <CR>MSH <CR>MSH <CR><NL></pre> <ul data-bbox="818 1528 1409 1698" style="list-style-type: none"> • len10 specifies a 10-byte record length, also called a length-encoded message. • eof specifies that the message has an end-of-file character, usually supplied by the computer's operating system.

Parameters	Description
CRNL Convert	<p>This creates any file on a Windows server as a Windows-type file and not UNIX-type. CRNL is for Windows and NL is for UNIX. This option is first performed on each message (record) in a file. After this, the message delimited Style is performed.</p> <p>CRNL to NL converts Windows-based files to UNIX-based files. NL to CRNL converts UNIX-based files to Windows-based files.</p>

TPS pane

This table shows TPS configuration parameters:

Parameters	Description
Directory Parse	<p>Use this TPS to select programmatically which files to process on each Scan Interval, and in what order. Click Edit to open the TPS Editor. Edit TPS Proc Properties in this dialog box.</p> <p>Note: Without a directory parse procedure on the inbound thread, the engine assumes that every file in the directory is to be processed.</p>
Deletion	<p>Click Edit to open the TPS Editor. This dialog box is used to select the TPS through which the driver passes the processed file names.</p>

Scheduling pane

This table shows scheduling configuration parameters:

Parameters	Description
Read Interval	<p>Specify the minimum time, in seconds, to wait between reading from a file.</p> <p>This tells the driver to process a message every <n> seconds as long as there is work to do (for example, there are input files available).</p> <p>In general, the read interval is fairly low so that the driver rapidly processes inbound messages.</p>
Max Messages	<p>Specify the maximum number of messages to read from the file on each Read Interval.</p>

Parameters	Description
Scan Interval	<p>Specify the minimum time, in seconds, between inbound directory scans for most message files.</p> <p>This is the (minimum) number of seconds between inbound directory scans to look for new work.</p> <p>This option is disabled if Use advanced scheduling is selected.</p> <p>For example, the Scan Interval is 6000 (10 minutes), the Read Interval is 10, and Max Messages is 30. The directory is scanned every 10 minutes for files. If files are found after a scan, then the first file is read every 10 seconds. 30 messages are retrieved every read until the file is exhausted. The next file in the list of files that is retrieved by the scan is read every 10 seconds with 30 messages every read, and so on. This is carried out until all the files from the current scan are read.</p>
Scan Mode	<p>For scheduling, there are two scanning options:</p> <ul style="list-style-type: none"> • Serial This option scans and picks the files in one of the IB directories in each interval. • Parallel This option scans all of the IB directories and then picks all of the files in each interval. <p>The scan order is the same as the directories order that is configured by the user.</p> <p>The default mode is serial mode. This mode is backwards compatible.</p> <p>Example:</p> <p>The directory list is: one, two, three, four.</p> <p>Only directory four contains a file.</p> <p>Read Interval is 5.</p> <p>Scan Interval is 20.</p> <p>In Serial mode, four scan cycles are completed to get to that file, for a total of 80 seconds.</p> <p>In Parallel mode, it takes 20 seconds.</p>
Use advanced scheduling	<p>Select this to schedule recurring events, such as cycling log or SMAT files. Then click Setup to open the Scheduling dialog box.</p>

Outbound pane

This table shows outbound configuration parameters:

Parameters	Description
Directory	Specify the local directory path. This specifies the directory for local file delivery.
File Name Template	<p>Fileset FTP stores the file name for IB files, and after routing to an outbound Fileset FTP, it uses the inbound file name by default. With this option, you can define the template for outbound file names.</p> <p>For example, if the template is <code>SYS.TIMESTAMP</code>, the outbound file name is similar to 20160411.</p> <p>In another example, an outbound file name of <code>site_testsite_p_5120.dat</code> has a template of <code>site_%CIS.SITE%_p_%MSG.PRIORITY%.dat</code>.</p> <p>If this field is left blank, then it reverts to the old behavior.</p> <p>If this directory is specified, then File and File Name Template cannot both be empty.</p> <p>For the templates, see Fileset FTP/Local outbound file name templates.</p>
File	Specify the default outbound file name.
Temporary File	Specify the path and name of the temporary file to use when transferring data. If this text box is empty, then no temporary file is used. When the transfer is successfully completed, the temporary file is renamed to the value of the <code>FTPTEMPFILE</code> key. If this key is empty, then the file is uploaded as the actual file name.

Parameters	Description
Style	<p data-bbox="818 302 1409 432">Click the arrow to select from the list how messages are stored within the files. This value is particularly important when numerous messages are written to a file.</p> <ul data-bbox="818 449 1409 583" style="list-style-type: none"> • If single, then the driver reads the contents of the file into a single message. • hl7 and nl specify different schemes for delimiting messages within the file. <p data-bbox="868 600 1409 699">For nl style, the delimiter is <NL> between messages. As each message ends with <CR>, between messages would look similar to <CR><NL>.</p> <p data-bbox="868 716 1019 747">For example:</p> <pre data-bbox="883 785 1393 810">MSH <CR><NL>MSH <CR><NL>MSH <CR><NL></pre> <p data-bbox="868 848 1409 982">File reads/writes in nl mode do not support the Unicode UTF-16 encoding. The nl mode cannot be performed without knowing the encoding beforehand, which is not known.</p> <p data-bbox="868 999 1409 1409">For hl7, the delimiter is <CR>MSH and the last character at end-of-file must be NL. The NL at the end-of-file indicates that it is the end of the message. Each message ends with <CR>. With the hl7 style there must be <CR>MSH in-between each message. The MSH is technically not a delimiter because it belongs to the next message. It serves only as the indicator for the beginning of the next message during parsing. There are no delimiters between HL7 messages, as the engine seeks the <CR>MSH to find the beginning of the next message.</p> <p data-bbox="868 1425 1019 1457">For example:</p> <pre data-bbox="883 1495 1318 1520">MSH <CR>MSH <CR>MSH <CR><NL></pre> <ul data-bbox="818 1558 1409 1732" style="list-style-type: none"> • len10 specifies a 10-byte record length, also called a length-encoded message. • eof specifies that the message has an end-of-file character, usually supplied by the computer's operating system.

Parameters	Description
Append data	<p>Click this to append data to the file, if it exists. Otherwise, the file is overwritten each time the thread starts up.</p> <p>Override all of these outbound control values with a message's DRIVERCTL metadata string.</p>

Start-Up Procedures pane

Click **Edit** to open the **TPS Editor**. Use this dialog box to select the procedures to run when the thread starts. This is a TPS (Tcl Procedure Stream). Use it to trade messages with the remote end.

Fileset FTP/Local outbound file name templates

Template name are consistent with those in Tcl commands. The %USERDATA% template gets the entirety of the USERDATA value. This can be empty or a long string.

The fileset protocols have a filter for inbound. This consists of a field to specify the file extension. For example, *.dat.

- White space is not replaced.
- The engine supports white spaces in the outbound (OB) file name. The full path length is restricted to 260.
- An error results if the limit is exceeded.

Templates from message metadata

- %MSG.TYPE%: msgType
- %MSG.CLASS%: msgClass
- %MSG.PRIORITY%: msgPriority
- %MSG.MID%: msgMid
- %MSG.HOSTID%: msgHostId
- %MSG.ORIGSOURCECONN%: msgOrigSrcThread
- %MSG.ORIGDESTCONN%: msgOrigDestThread
- %MSG.SOURCECONN%: msgSrcThread
- %MSG.DESTCONN%: msgDestThread
- %MSG.TRXID%: DRIVERCTL. _TRXID_
- %MSG.TIMEIN%: msgTimeStartIb
- %MSG.TIMEOUT%: msgTimeStartOb
- %MSG.USERDATA%: msgUserData

Additional templates

- %SYS.TIME%
The time of writing OB files. For example, 09-10-YYYY.

- `%SYS.DATE%`
The date of writing OB files. For example, 03-31-YYYY.
- `%SYS.TIMESTAMP%`
The timestamp of writing OB files.
- `%CIS.SEQUENCE%`
Starts from 0 and increases by one (1) for every OB message.
To keep the state of the sequence after a process stop:
 - If you started the whole process, then the sequence is set to 0.
 - If you start only an individual thread, then after restart the sequence number continues from the last sequence number.
- `%CIS.IBFILENAME%`
The inbound (IB) file name read from `DRIVERCTL.FILENAME`.
- `%CIS.PROCESS%`
The process name.
- `%CIS.SITE%`
The site name.
- `%CIS.THREAD%`
The thread name.
- `%CIS.HOSTNAME%`
The host name.

Notes:

- If the template is `%USERDATA%.dat` and the `USERDATA` is empty, then the OB file name is `.dat`.
- If the value of the template is empty or includes characters that cannot be used as a file name, then an error is reported for an invalid OB file name. For example, `*:?"<>`.

Fileset FTP protocol: cURL Options tab

The **cURL Options** tab consists of a table composed of key-value pairs for cURL options.

This table contains columns for Name, or Key, and Value. Every row in the table is a key-value pair for a cURL option.

Supported cURL parameters are:

- `long`
- `object pointer` (only includes a `char *` to a zero-terminated string)
- `curl_off_t`

cURL does not support the `function pointer` parameter and other object pointers, for example, `FILE *`.

If a user-defined cURL option conflicts with one that the system has used, then the user-defined option overwrites the existing one.

If you define the same cURL option twice, then the latter one overwrites the former one.

To help in performing cURL-related testing before configuration, a cURL command-line tool can be found in `$HCIRoot/bin`.

Supported cURL options

This table shows the supported cURL options:

Option	Option	Option	Option
CURLOPT_APPEND	CURLOPT_FTP_RESPONSE_TIMEOUT	CURLOPT_MAX_SEND_SPEED_LARGE	CURLOPT_SSH_PUBLIC_KEYFILE
CURLOPT_AUTOREFERER	CURLOPT_FTP_SKIP_PASV_IP	CURLOPT_MAXFILE_SIZE_LARGE	CURLOPT_SSL_VERIFY_HOST
CURLOPT_CAINFO	CURLOPT_FTP_USE_EPRT	CURLOPT_MAXFILESIZE	CURLOPT_SSL_VERIFYPEER
CURLOPT_CAPATH	CURLOPT_FTP_USE_EPSV	CURLOPT_MAXREDIRS	CURLOPT_SSLCERTTYPE
CURLOPT_CERTINFO	CURLOPT_FTPPORT, 10017,	CURLOPT_NETRC_FILE	CURLOPT_SSENGINE_DEFAULT
CURLOPT_CONNECT_TIMEOUT_MS	CURLOPT_HEADER	CURLOPT_NETRC	CURLOPT_SSENGINE
CURLOPT_CONNECT_TIMEOUT	CURLOPT_HTTP_CONTENT_DECODING	CURLOPT_NEW_DIRECTORY_PERMS	CURLOPT_SSLKEY
CURLOPT_COOKIE	CURLOPT_HTTP_TRANSFER_DECODING	CURLOPT_NEW_FILE_PERMS	CURLOPT_SSLKEYTYPE
CURLOPT_COOKIEFILE	CURLOPT_HTTP_VERSION	CURLOPT_PASSWORD	CURLOPT_SSLVERSION
CURLOPT_CRLF"CURLOPT_CRLF"	CURLOPT_HTTPAUTH	CURLOPT_PORT	CURLOPT_TCP_NODELAY
CURLOPT_CUSTOMREQUEST	CURLOPT_IGNORE_CONTENT_LENGTH	CURLOPT_PROXY_TRANSFER_MODE	CURLOPT_TIMEOUT_MS
CURLOPT_DIRLISTONLY	CURLOPT_INTERFACE	CURLOPT_PROXY	CURLOPT_TIMEOUT
CURLOPT_DNS_CACHE_TIMEOUT	CURLOPT_ISSUERCERT	CURLOPT_PROXYPASSWORD	CURLOPT_TRANSFERTEXT
CURLOPT_FAILONERROR	CURLOPT_KEYPASSWD	CURLOPT_PROXYPORT	CURLOPT_URL
CURLOPT_FILETIME	CURLOPT_LOCALPORT	CURLOPT_PROXYTYPE	CURLOPT_USERAGENT

Option	Option	Option	Option
CURLOPT_FOR-BID_REUSE	CURLOPT_LOCALPORTRANGE	CURLOPT_PROXYUSERNAME	CURLOPT_USERNAME
CURLOPT_FRESH_CONNECT	CURLOPT_LOW_SPEED_LIMIT	CURLOPT_PROXYUSERPWD	CURLOPT_USERPWD
CURLOPT_FTP_ACCOUNT	CURLOPT_LOW_SPEED_TIME	CURLOPT_REFERER	CURLOPT_VERBOSE
CURLOPT_FTP_CREATE_MISSING_DIRS	CURLOPT_MAX_RECV_SPEED_LARGE	CURLOPT_SSH_PRIVATE_KEYFILE	

Fileset FTP protocol: FTP Options tab

When the driver connects to the remote server, or must change its connection to the server, it sends commands from these values.

This table shows the FTP account information to specify.

Note: You can use absolute paths. These must begin with %2F or /.

Field	Description
Login	Specifies the user name to log in to the server.
Password	Specifies the password to log in to the server.
Account Info	Specifies any account information necessary to log in to the server. If the server does not prompt for the password or the account information, then they are not sent. Refer to your FTP server documentation to determine what is required to log in.

Having logged on, the driver sets the default transfer type.

- If the log-in process is prompted by an inbound or outbound operation, then further commands to that end are sent.
- If the driver is initializing itself, then it sends no further commands until required.

FTP Options pane

This tables shows FTP Options pane parameters. When the driver connects to the remote server, or must change its connection to the server, it sends these commands, as required:

Parameter	Description
Local Binding Address	Specify the IP address or a resolvable host name, or click List to make a selection.

Parameter	Description
Response Timeout	<p>Specify the number of seconds the driver should wait for a response after a command is sent.</p> <p>When the server's response arrives, the driver reads, parses, and handles the response, continuing the operation it started.</p> <p>If it does not receive a response, then the driver yields back to the command thread so that other threads can run.</p>
Dir List Command	Specify a string that looks for inbound files in the remote inbound directory.
Close connection after write	Select this to close the connection after every outbound write. Clear this to keep the connection open. Usually, the FTP connection is open.
Delay connection until needed	Select this to delay the connection until it is required. Clear this to make the connection during thread start-up.
Data Type	Specify the default transfer type.
ACTIVE Mode	<p>In ACTIVE mode, the FTP client opens a dynamic port. It then sends the FTP server the dynamic port number on which it is listening over the control stream. At this point, waits for a connection from the FTP server. When the FTP server initiates the data connection to the FTP client, it binds the source port to port 20 on the FTP server.</p> <p>In passive mode, when the check box is cleared, the FTP server opens a dynamic port. It then sends the FTP client the server's IP address. This is the address to connect with and the port on which to listen over the control stream. Then it waits for a connection from the FTP client. This is a 16-bit value that is broken into a high and low byte. In this case, the FTP client binds the source port of the connection to a dynamic port.</p>
Secure Option	Use the to select the type of secure implementation and then click Configure to configure the settings.
Max Retries	This is the maximum number of retries when Fileset FTP encounters an error. This is supported in CLAPI.

Most of the FTP configuration applies to all inbound and outbound FTP operations. The driver maintains one FTP connection at a time. If the FTP connection is lost or times out, then the driver attempts to re-establish it. This happens if, or when, it is required for the next FTP operation.

FTP Host Information pane

This table shows FTP host information parameters:

Parameter	Description
Host	The machine with which to connect. Click List to open the Hosts List box. Select a host name and click Apply .
Port	The port/service with which to connect. Click List to open the Ports List box. Select a port name and click Apply .

FTP Protocol Start-Up Procedures pane

Click **Edit** to open the **TPS Editor**. Use this dialog box to select the procedures to run when the thread starts. This is a TPS (Tcl Procedure Stream). Use it to trade messages with the remote end.

Configuration details

The driver initializes itself after it has internalized its configuration. If `MODE` is `ftp` and the `FTPDELAY` flag is not set, then the driver attempts to connect to the configured FTP server and log in.

If the driver is configured for inbound, then it initializes that.

- If `IBSTYLE` is anything except `single`, then it initializes its multi-message state by opening its multi-message state file, `fileset-<threadname>`.
- If that file shows the driver was in the midst of processing a file when it shut down, then the driver opens the data file and prepares to read from the correct position. It does this by skipping past messages it has already read.
- If `MODE` is `"local-oldest"` or `"local-tps"`, then the driver tests to ensure it can open and access `IBDIR`.
- If `MODE` is `"local-tps"` or `"ftp"`, then the driver runs the `IBDIRPARSE` and `IBDEL` TPS procedures in start-up mode.

`CONTINUE` messages returned by each are handled as usual. There is no outbound initialization.

`MODE` determines locality, or file transfer directions, for all of its operations. How the driver determines whether it is configured for inbound-only, inbound and outbound, or outbound-only depends on the `MODE`.

`MODE` choices are:

- The `"local-oldest"` mode scans the inbound directory and builds an oldest-first list of the files. It re-scans the directory if it was accessed or modified from the last scan. As long as there are files available, it processes them in order according to `IBSTYLE`. When it completes each file, it removes it from the inbound directory.
- The `"local-tps"` and `"ftp"` modes rely on user-supplied TPS procedures to parse inbound directory listings and populate an ordered list of files to process. When the driver's list is empty, it scans the inbound directory and places the results into a message.
- The `"local-tps"` mode lists space-separated file names from the inbound directory, which skips only `"."` and `".."`.

- The "ftp" mode, if required, sends the `CWD <FTP_IBDIR>` command to the server to move into the correct directory. Then, send the `FTPDIRLISTCMD` string. Specify a `FTPDIRLISTCMD` string that generates the data your procedure requires to order the files.

The driver reads and processes inbound files and messages if the `IBDIR` value is not empty for `local-*`, or the `FTP_IBDIR` value is not empty for `ftp`. It similarly processes outbound operations using the `OBDIR` and `FTPOBDIR` values.

If `MODE` is "ftp" or "local-tps", then the driver uses the `IBDIRPARSE` and `IBDEL` TPS UPoCs to determine the order that available files are processed.

Specify "ftp" if the driver is to perform FTP operations.

Specify "local-oldest" or "local-tps" for local-only operation.

Inbound configuration

In general, the driver has two phases of inbound operation, where it identifies inbound files and then processes the inbound files.

The driver performs its inbound duties under timer control. The specifics of what the driver does when the timer counts down depends on `MODE`.

Many FTP servers support the `NLST` and `LIST` commands. Consult your FTP server documentation to configure the `FTPDIRLISTCMD` string.

The driver passes the message into the `IBDIRPARSE` TPS.

This table shows the dispositions handled by the driver:

Disposition	Action
ERROR	Places the message in the error database.
KILL	Destroys the message.
OVER	Illegal; destroys the message.
PROTO	Places the message on the OB-Post-TPS queue.
SEND	Places the message on the IB-Post-TPS queue.

`CONTINUE` messages are ordered lists of files to process. The driver does not perform another inbound scan or run the `IBDIRPARSE` TPS until the list is empty.

When the driver finishes processing a file, it places the file's name in a message and passes it through the `IBDEL` TPS. Non-`CONTINUE` dispositions are handled as above; `CONTINUE` messages are parsed as ordered lists of files to delete.

- If `local-tps`, then the driver removes those files from the inbound directory.
- If `ftp`, then the driver sends a `DELE file name` command to the FTP server.

As long as the driver has inbound files to process, it processes a message according to `IBREADINTERVAL`.

The driver reads single-message files into a message and passes them into the engine. For FTP, the driver contacts the remote server, receives the data into a message, and passes it into the engine.

If configured for multi-message files, then the driver receives the file data from the FTP server and writes it to a local file (<threadname>-FTP-temp). After this is on the local disk, it is treated as a local multi-message file. The driver reads the next message, passes it into the engine, and updates the multi-message recovery state file.

When the driver has exhausted the list of input files, it reschedules itself for `IBSCANINTERVAL` seconds. As long as there are no files available (the inbound directory is empty for local-oldest or the `IBDIRPARSE` TPS produces no, or empty, `CONTINUE` messages), the driver rescans every `IBSCANINTERVAL` seconds.

Outbound configuration

The Fileset FTP driver writes messages to its connection by writing to files. Before writing a message, the driver examines the message's `DRIVERCTL` metadata string. If it contains a `FILESET` key, then it takes that key's value as a configuration list for overriding the default configuration list.

For example:

```
{ APPC { ... <APPC control> }} { FILESET {<Fileset FTP control> }} ...
```

The control list is a keyed list, using many of the same keys as the NetConfig list:

- FTPACCT
- FTPCLOSE
- FTPHOST
- FTPOBDIR
- FTPPASSWD
- FTTPORT
- FTPTYPE
- FTPUSER
- OBAPPEND
- OBDIR
- OBFIL
- OBSTYLE

Note: The `DRIVERCTL` string cannot override the driver's `MODE` configuration. You cannot use `DRIVERCTL` to force the driver to deliver one message locally when `MODE` is `ftp` or vice-versa.

Other examples are:

To append the newline-terminated data to `obdata.123`:

```
{ FILESET {
{ OBAPPEND 1 } { OBFIL obdata.123 } { OBSTYLE n\ }
}}
```

To use `newacct` account information for the current user, create/overwrite `data321.dat` using image/binary transfer. Then, close the connection when the transfer is complete:

```
{ FILESET {
{ FTPACCT newacct } { FTPCLOSE 1 } { FTPTYPE i }
{ OBAPPEND 0 } { OBFIL data321.dat }
}}
```

A message's `DRIVERCTL` values apply only to the current message. They do not affect inbound flow or other outbound messages.

As the driver writes the outbound message, it includes message-delimiter characters appropriate to `OBSTYLE`.

Local delivery

If `OBFIL` is a relative path, then it is relative to `OBDIR`. This is relative to the process binary directory.

If the `OBAPPEND` flag is set, then the driver opens `OBFIL` in append mode. Otherwise, it truncates and overwrites the file.

FTP

If `FTPHOST` specifies another `FTPHOST/FTPUSER/FTPACCT/FTPTYPE` configuration than it currently has, then it adjusts its connection. This could even be to the point of connecting to a new server.

If not in the correct directory, then it sends the `CWD FTPOBDIR` command to move to the correct directory.

If the `OBAPPEND` flag is set, then the driver sends the `APPE OBFIL` command to append the message data to the remote file. Otherwise, it sends `STOR OBFIL` to create/overwrite the remote file.

Socket Local IP binding

Use the **Local Binding Address** feature to control which local IP address is used by the system sockets. For example, in high availability systems where a standby host takes over the workload of a failed host. In this case, the standby host would be indistinguishable from the failed host. That is, the local IP address of the standby host uses the IP address of the failed host.

If no local binding is specified in the dialog box, then client sockets are not bound. The operating system assigns any valid IP address to the local end of a socket. Socket Local IP Binding, though, can control which IP address is used for the local end of each socket.

Capabilities are:

- Client sockets can be made to originate from any valid IP address on the machine, giving the ability to get through firewalls.
- Multiple servers can listen on the same port as long as they are bound to different local IP addresses.
- This capability is available for Fileset FTP, HTTP Client, PDL-TCP/IP and TCP/IP.
- PDL TCP/IP and TCP/IP have the further capability to permit local IP binding of server and multi-server sockets.
- If no local binding is specified, then there is no local binding for clients and server sockets are bound to `IN_ADDR_ANY`.

- The local binding can be specified as an IP address or a resolvable host name.

Configuration keys

This table shows the configuration keys and their types:

Key	Type	Description
FTPACCT	string	Account info string for FTP user
FTPCLOSE	0 or 1	Close FTP connection after write
FTPDELAY	0 or 1	Delay FTP connect until required
FTPDIRLISTCMD	string	FTP directory list command
FTPHOST	string/address	FTP server name/address
FTPIBDIR	string	Remote IB directory
FTPOBDIR	string	Remote OB directory
FTPPASSWD	string	FTP user's password
FTPPORT	string/numeric	FTP server port/service
FTPPULSETIME	numeric	NOOP heartbeat interval
FTPRESPONSETIMEOUT	numeric	Blocking-response time-out
FTPTYPE	a: ASCII i: image/binary	FTP transfer type
FTPUSER	string	FTP user log in
IBDIR	string	Local IB directory
IBDEL	TPS	IB deletion TPS
IBDIRPARSE	TPS	IB directory parse TPS
IBREADINTERVAL	numeric	IB read interval
IBSCANINTERVAL	numeric	IB scan interval
IBSTYLE	hl7: HL7 (multi-message) nl: Newline-terminated (multi-message) single: Single message	IB file style

Key	Type	Description
MODE	ftp: FTP IB or OB local-oldest: Local files; oldest-first IB local-tps: Local files; TPS-ordered IB	Operating mode
OBAPPEND	0 or 1	Append OB data
OBDIR	string	Local OB directory
OBFILE	string	Default OB file name
OBSTYLE	hl7: HL7 len10: 10-byte length-encoded nl: Newline-terminated single: Single message	OB file style
TYPE	"fileset"	Identifies this driver

Configuring security

Secure Option options are:

- Blank: No security.
- FTPS: Adds support for cryptographic protocols.
- SFTP: An additional means of securely transferring files across the Internet, providing the ability to have secure, efficient, file transfer. This feature requires an enterprise license.

FTPS

Select **FTPS** from the menu and click **Configure** to open the **FTPS** dialog box, where SSL-related settings are configured.

Note: TLS session resumption is disabled when a client certificate is used. When connecting to an FTP server when a client certificate is used, ensure **TLS Session Resumption** is not enabled on the FTP server side.

- Encryption options are:
 - **Implicit:** This is implicit encryption, where the client connects to an implicitly secure port on the server.
 - **Explicit:** This is explicit encryption, where the client issues FTP commands to explicitly tell the server to go to secure mode. This is the default.
- **Mode**
Select the mode from the list.
- **SSL Protocol**

This is used to select the openssl version from the list: All, TLSv1, TLSv1.1, TLSv1.2, and TLSv1.3. When you select an SSL protocol, a description of the selected protocol is shown in the comment field together with the Mode description.

- **SSL Cipher Suites**

Set ciphers in this field. If no cipher is set, then the default cipher suites are used. If this field is not set, then the default cipher suites are used.

- Anonymous mode:
server: !DEFAULT:HIG:ADH
Client: ALL
- Non-anonymous mode:
HIGH:RC4+RSA:+MD5:!DHE:!3DES:!EXP:!ADH:!AES256-SHA:!AES128-SHA:!EDH:!aNULL:!eNULL:!NULL

SFTP

Select the **SFTP** option from the menu and click **Configure** to open the **SFTP** dialog box, where SSL-related settings are configured.

The Private Key **Password** field is a password and is encrypted in NetConfig.

Note:

- An SFTP server using OpenSSH does not support the append option on SFTP uploads. When using the append option on SFTP uploads, the server must support this option.
- `libcurl` does not do a complete ASCII conversion when doing ASCII transfers over FTP. This is a known limitation. `libcurl` sets the mode to ASCII and performs a standard transfer. Therefore, be careful when using ASCII mode in SFTP transfers.

Generating the private key/public key/password

You can use the `ssh-keygen` utility to generate the private and public keys.

This is a UNIX utility that is used to generate, manage, and convert authentication keys for SSH authentication.

The `ssh-keygen` utility is part of the UNIX/Linux install. The system SFTP protocol is an SFTP client that can connect to a specific SFTP server by configuration.

- On UNIX/Linux, the `ssh` daemon service (SFTP server) is generally installed by default.
- On Windows, you must install third-party SFTP server software to have the system SFTP protocol to connect to the Window's SFTP server.

Different SFTP server applications have their own configurations. Refer to the third-party software documentation for details.

A system SFTP protocol running on Windows can connect to a UNIX/Linux SFTP server. Installing SFTP server software on Windows is not required, but based solely on your particular deployment.

This is an example for generating a DSA type private/public key using Linux.

- 1 Log in to a Linux box with `hci` user.

2 Generate a dsa type key. For example:

```
[hci@hvsheuer ~]$ ssh-keygen -t dsa
Generating public/private dsa key pair.

Specify the file in which to save the key (/home/hci/.ssh/id_dsa):
Specify passphrase (empty for no passphrase):
Specify same passphrase again:

Your identification has been saved in /home/hci/.ssh/id_dsa.
Your public key has been saved in /home/hci/.ssh/id_dsa.pub.

The key fingerprint is:
d9:ca:fd:ff:cb:f7:68:b6:cd:0c:9f:c5:4c:20:a3:b6 hci@hvsheuer
```

By default the generated private/public key is saved under the `~/.ssh/` folder.

`id_dsa` is the private key

`id_dsa.pub` is the public key

Note: Although the private/public key pair is generated on a Linux box, it can be also used with other platforms. For example:

```
[hci@hvsheuer ~]$ ls -al ~/.ssh
total 32
drwx----- 2 hci staff 4096 Mar 16 12:33 .
drwxrwxr-x 18 hci staff 4096 Mar 16 12:32 ..
-rw----- 1 hci staff 736 Mar 16 12:33 id_dsa
-rw-r--r-- 1 hci staff 602 Mar 16 12:33 id_dsa.pub
```

Using the generated private/public key pair: SFTP client side

1 Copy the key pair to the SFTP client side.

Place them in a specific folder, for example, `$HCIRoot/sftptestsite/`. You can also rename the key file name if required.

2 Configure the full path of the private/public key file in the **SFTP** dialog box. This is **Secure Option > SFTP > Configure**.

Using the generated private/public key pair: SFTP server side (UNIX)

1 Ensure the SSH daemon service is enabled.

2 Log in with hci user and create a new folder `.ssh` under the `/home/hci/` folder.

```
[hci@hvsheuer ~]$ mkdir .ssh
```

3 Change the permission of `~/.ssh`, `~/.ssh/.`, `~/.ssh/..` to 700. That is, `drwx-----`.

```
[hci@hvsheuer ~]$ su
Password:[root@hvsheuer hci]# chmod 700 /home/hci/.ssh
[root@hvsheuer hci]# cd /home/hci/

[root@hvsheuer hci]# chmod 700 /home/hci/.ssh/.
[root@hvsheuer hci]# chmod 700 /home/hci/.ssh/..
```

```
[root@hvsheuer hci]# ls -al
drwx----- 2 hci staff 4096 Mar 16 14:03 .ssh
[root@hvsheuer hci]# ls -al .ssh
total 16
drwx----- 2 hci staff 4096 Mar 16 14:03 .
drwx----- 18 hci staff 4096 Mar 16 14:03 ..

[root@hvsheuer .ssh]# exit
exit
[hci@hvsheuer ~]$
```

- 4 Create a new `authorized_keys` file under `/home/hci/.ssh` and copy the public key content to `/home/hci/.ssh/authorized_keys`.

```
cat id_dsa.pub >> /home/hci/.ssh/authorized_keys
```

- 5 Ensure the permission of file `authorized_keys` is `-rw-r--r--`.

SFTP server configuration on Windows

To configure the SFTP server on Windows, refer to your SFTP Server tool documentation.

File protocol

The File protocol driver reads messages from one file and writes them to another. The driver supports both newline-terminated and 10-byte length-encoded messages.

File is usually used in the testing phase as a sample data file that simulates messages sent across a connection. To test an inbound connection, populate the file with test data before testing. This is data that was captured from the network or individual test messages. To test an outbound connection, save the test data to a file.

Best practices:

- Guarantee a minimum delay between message reads.
- Overwrite the outbound file, if it already exists, or append new data to it.

Input pane

This table shows input configuration options:

Option	Description
Filename	<p>This names the input file to use and can contain multiple messages. Click Browse to open a file browser.</p> <p>Use the name of the simulated data file from which input data is read. If testing an outbound thread, then leave the text field blank, or specify the null device (<code>nul:</code> on Windows; <code>/dev/null</code> on UNIX).</p> <ul style="list-style-type: none">• If an absolute pathname is not specified, then the pathname is relative to the binary directory for the process running under this driver.• If a file name is not specified, then the engine uses <code>/dev/null</code> by default, and the driver does not produce any inbound messages. <p>You must specify a file name in the Input Filename or Output Filename field. Both fields cannot be blank.</p>
CRNL Convert	<p>This creates any files on a Windows server as Windows-type files and not as UNIX-type files. CRNL is for Windows and NL is for UNIX. This option is performed on each message (record) in a file.</p> <ul style="list-style-type: none">• CRNL to NL converts Windows-based files to UNIX-based files.• NL to CRNL converts UNIX-based files to Windows-based files.
File Format	<p>The driver identifies message boundaries according to the selected file format:</p> <ul style="list-style-type: none">• Newline terminated ends every message with a newline character (for example, <code>\n</code>).• Length-encoded is 10-byte length-encoding which prepends 10 ASCII numeric characters, denoting the message length before each message (for example, <code>0000000007message</code>).• End-of-file terminated ends every message with an end-of-file character.

Option	Description
Input Delay	<p>This specifies the minimum number of seconds between reading messages from the input file.</p> <p>As long as the file contains data to read, the driver reads the next message into the engine.</p> <p>To read messages rapidly, leave the value at 0. Use a larger number to delay each subsequent read to simulate occasional delivery or to give other threads more opportunity to run.</p> <p>When the driver reaches the end-of-file (EOF), it enters the ineof protocol state. Until the thread restarts, it does not read any more messages. This does not affect outbound data flow.</p>

Output pane

This table shows output configuration options:

Option	Description
Filename	<p>This names the outbound message file. Specify the name of the file where output data is stored. Click Browse to open a file browser.</p> <p>If testing an inbound thread, then leave the text field blank. Similar to the input Filename, it is relative to the process binary directory unless it is an absolute pathname. The process binary directory is the current working directory for the process when it runs.</p> <p>Specify a file name in the Input Filename or Output Filename field. Both fields cannot be blank.</p>
CRNL Convert	<p>This creates any files on a Windows server as Windows-type files and not as UNIX-type files. CRNL is for Windows and NL is for UNIX. This option is performed on each message (record) in a file.</p> <ul style="list-style-type: none"> • CRNL to NL converts Windows-based files to UNIX-based files. • NL to CRNL converts UNIX-based files to Windows-based files.

Option	Description
File Format	<p>The driver identifies message boundaries according to the selected file format:</p> <ul style="list-style-type: none"> • Newline terminated ends every message with a newline character (for example, \n). • Length-encoded is 10-byte length-encoding which prepends 10 ASCII numeric characters, denoting the message length before each message (for example, 0000000007message). • End-of-File terminated ends every message with an end-of-file character.
Append to file	<p>This determines what action the driver takes on start-up when an outbound file exists.</p> <p>When this is cleared, the outbound message does not append to the output file but instead overwrites the output file.</p>

NL to CRNL conversion

The File protocol does not convert NL to CRNL if the data has 0A and 0D 0A. Before the conversion, the engine checks if CRNL already exists to determine if the data has already been processed.

Any CRNL that exists in the data causes the engine to stop the conversion. This is by design. Otherwise, CRNL is converted to CRNLNL.

File Start-Up Procedures pane

Click **Edit** to open the **TPS Editor**. Then, select the procedures to run when the thread starts. This is a Tcl Procedure Stream (TPS). Use it to trade messages with the remote end.

Fileset/FTP Local protocol

Fileset Local handles groups (directories) of files locally by FTP. The driver can process files containing multiple messages, saving state to continue at the correct point across stop/start boundaries.

Note: Fileset-FTP and Fileset Local populate DRIVERCTL metadata with the Inbound file name as a keyed list entry {FILENAME xxx}.

Fileset/FTP Local Inbound pane

This table shows inbound parameters:

Parameter	Description
Directory	Specify the inbound file directory path (IBDIR). Click Browse to open a file browser to search for the directory. You can also specify multiple directories. See Scanning multiple directories for inbound pick-up .
File Mask, Regular Expression, Case Sensitive	See Fileset mask .

Parameter	Description
Style	<p>Click the arrow to select from the list how messages are stored within the files (IBSTYLE). This determines how the driver processes each file.</p> <ul style="list-style-type: none"> If single, then the driver reads the contents of the file into a single message. hl7 and nl specify different schemes for delimiting messages within the file. <p>For nl style, the delimiter is <NL> between messages. As each message ends with <CR>, between messages would look similar to <CR>.</p> <p>For example:</p> <pre>MSH <CR>MSH <CR>MSH <CR></pre> <p>File read/writes in nl mode do not support the Unicode UTF-16 encoding. The nl mode cannot be performed without knowing the encoding beforehand, which is not known.</p> <p>For hl7, the delimiter is <CR>MSH and the last character at end-of-file must be NL. The NL at the end-of-file indicates that it is the end of the message. Basically, each message ends with <CR>. With the hl7 style there must be <CR>MSH in-between each message. The MSH is technically not a delimiter because it belongs to the next message. It serves only as the indicator for the beginning of the next message during parsing. In this sense, there are no delimiters between HL7 messages, as the engine seeks the <CR>MSH to find the beginning of the next message.</p> <p>For example:</p> <pre>MSH <CR>MSH <CR>MSH <CR></pre> <ul style="list-style-type: none"> len10 specifies a 10-byte record length, also called a length-encoded message. eof specifies that the message has an end-of-file character, usually supplied by the computer's operating system.

Parameter	Description
CRNL Convert	<p>This creates any files on a Windows server as Windows-type files and not as UNIX-type files. CRNL is for Windows and NL is for UNIX. This option is first performed on each message (record) in a file. After this, the message delimited Style is performed.</p> <p>CRNL to NL converts Windows-based files to UNIX-based files. NL to CRNL converts UNIX-based files to Windows-based files.</p>
Order	<p>Select this to specify the order in which files are picked up.</p> <p>Alpha picks up files in alphabetic order.</p> <p>Timestamp picks up files in time stamp order (default).</p>

Fileset/FTP Local TPS pane

This table shows TPS parameters:

Parameter	Description
TPS	<p>Select this to use directory parse or deletion TPS procs.</p> <p>When this box is cleared, the Directory Parse and Deletion options are unavailable.</p>

Parameter	Description
Directory Parse	<p>Use this TPS to select programmatically which files to process on each Scan Interval, and in what order. Click Edit to open the TPS Editor. Edit TPS proc properties in this dialog box.</p> <p>In the Directory Parse TPS Editor dialog box, List Full Directory is a special flag for the engine to pass a full directory list to TPS. By default, this is cleared. When this is selected, you must modify the TPS code to handle the list and return a name-only list back to the engine.</p> <p>When this is selected, the <code>msg_list</code> that is passed to the name parse TPS contains a full list, not only file names.</p> <p>This list is similar to <code>{{filename1 filesize mtime}{filename2 size mtime}... ..}</code>.</p> <p>To get a full list from FTP, use <code>list</code> instead of the default <code>nlist</code>. The format of the full list is similar to the result of <code>ls -l</code>, depending on the FTP servers.</p> <p>Note: When this option is selected, using the default directory parse TPS causes an engine error or panic.</p>
Deletion	<p>Click Edit to open the TPS Editor. Use this dialog box to select the TPS through which the driver passes processed file names.</p>

Fileset/FTP Local Scheduling pane

This table shows scheduling parameters:

Parameter	Description
Read Interval	<p>Specify the minimum time, <code>IBREADINTERVAL</code>, in seconds, to wait between reading from a file.</p> <p>This tells the driver to process a message every <code>IBREADINTERVAL</code> seconds as long as there is work to do. For example, when there are input files available.</p> <p>In general, <code>IBREADINTERVAL</code> is fairly low so that the driver rapidly processes inbound messages.</p>
Max Messages	<p>Specify the maximum number of messages to read from the file on each read interval.</p>

Parameter	Description
Scan Interval	<p>Specify the minimum time <code>IBSCANINTERVAL</code>, in seconds, between inbound directory scans for most message files.</p> <p>This is the (minimum) number of seconds between inbound directory scans to look for new work.</p> <p>This option is disabled if Use advanced scheduling is selected.</p>
Scan Mode	<p>For scheduling, there are two scanning options:</p> <ul style="list-style-type: none"> • Serial This option scans and picks the files in one of the IB directories in each interval. • Parallel This option scans all of the IB directories and then picks all of the files in each interval. <p>The scan order is the same as the directories order that is configured by the user.</p> <p>The default mode is serial mode. This mode is backwards compatible.</p> <p>Example: The directory list is: one, two, three, four. Only directory four contains a file. Read Interval is 5. Scan Interval is 20.</p> <p>In Serial mode, four scan cycles are completed to get to that file, for a total of 80 seconds. In Parallel mode, it takes 20 seconds.</p>
Use advanced scheduling	<p>Select this to schedule recurring events, such as cycling log or SMAT files.</p> <p>Then, click Setup to open the Scheduling dialog box.</p>

Fileset/FTP Local Outbound pane

This table shows outbound parameters:

Parameter	Description
Directory	Specify the local directory path (<code>OBDIR</code>). This specifies the directory for local file delivery.

Parameter	Description
File Name Template	<p>Fileset Local stores the file name for IB files, but after routing to an outbound Fileset FTP, it uses the inbound file name by default. With this option, you can define the template for outbound file names.</p> <p>For example, if the template is <code>SYS.TIMESTAMP</code>, the outbound file name is similar to 20160411.</p> <p>In another example, an outbound file name of <code>site_testsite_p_5120.dat</code> has a template of <code>site_%CIS.SITE%_p_%MSG.PRIORITY%.dat</code>.</p> <p>If this field is left blank, then it reverts to the old behavior.</p> <p>If this directory is specified, then File and File Name Template cannot both be empty.</p> <p>For the templates, see Fileset FTP/Local outbound file name templates on page 661.</p>
File	<p>Specify the default outbound file name (<code>OBFILE</code>).</p>
Temporary File	<p>Specify the path and name of the temporary file to use when transferring data. If this text box is empty, then no temporary file is used. When the transfer is successfully completed, the temporary file is renamed to the value of the <code>FTPTEMPFILE</code> key. If this key is empty, then the file is uploaded as the actual file name.</p>

Parameter	Description
Style	<p>Click the arrow to select from the list how messages are stored within the files (OBSTYLE). This value is particularly important when numerous messages are written to a file.</p> <ul style="list-style-type: none"> If single, then the driver reads the contents of the file into a single message. hl7 and nl specify other schemes for delimiting messages within the file. <p>For nl style, the delimiter is <NL> between messages. As each message ends with <CR>, between messages would look similar to <CR><NL>.</p> <p>For example:</p> <pre>MSH <CR><NL>MSH <CR><NL>MSH <CR><NL></pre> <p>File read/writes in nl mode do not support the Unicode UTF-16 encoding. The nl mode cannot be performed without knowing the encoding beforehand, which is not known.</p> <p>For hl7, the delimiter is <CR>MSH and the last character at end-of-file must be NL. The NL at the end-of-file indicates that it is the end of the message. Basically, each message ends with <CR>. With the hl7 style there must be <CR>MSH in-between each message. The MSH is technically not a delimiter because it belongs to the next message. It serves only as the indicator for the beginning of the next message during parsing. In this sense, there are no delimiters between HL7 messages, as the engine seeks the <CR>MSH to find the beginning of the next message.</p> <p>For example:</p> <pre>MSH <CR>MSH <CR>MSH <CR><NL></pre> <ul style="list-style-type: none"> len10 specifies a 10-byte record length, also called a length-encoded message. eof specifies that the message has an end-of-file character, usually supplied by the computer's operating system.

Parameter	Description
Append data	<p>Select this to append data to the file (OBAPPEND), if it exists. Otherwise, the file is overwritten each time the thread starts up.</p> <p>Override all of these outbound control values with a message's</p> <p>DRIVERCTL</p> <p>metadata string.</p>
CRNL Convert	<p>This creates any files on a Windows server as Windows-type files and not as UNIX-type files. CRNL is for Windows and NL is for UNIX. This option is first performed on each message (record) in a file. After this, the message delimited Style is performed.</p> <p>CRNL to NL converts Windows-based files to UNIX-based files. NL to CRNL converts UNIX-based files to Windows-based files.</p>

Fileset/FTP Local Start-Up Procedures pane

Click **Edit** to open the **TPS Editor**. Use this dialog box to select the procedures to run when the thread starts. This is a TPS (Tcl Procedure Stream). Use it to trade messages with the remote end.

Fileset mask

The Fileset protocol supports the use of "*" as a wildcard for inbound file names. Specifying a file mask, for example, *.dat in **File Mask** configures the Fileset inbound protocol to process only files that match the specified mask.

The Inbound pane of the **Fileset/FTP Local Protocol Properties** dialog box has a **File Mask** field. Specifying a value configures the protocol to process only files that match the mask. If this field is blank, then all found files are processed.

When **Case Sensitive** is selected, the file mask is treated as case sensitive. the default is cleared.

When **Regular Expression** is selected, the engine processes the file mask as a regular expression instead of a basic pattern.

When this is selected, any text that is specified in **File Mask** is validated for correct regular expression syntax. If the validation fails, then the user is notified and the configuration cannot be saved.

If **Regular Expression** is selected, then **Case Sensitive** is disabled.

Protocol settings are stored in NetConfig. Debug logging displays a list of files found that matches the file mask.

Configuration

To specify a file mask, configure the item in NetConfig with the IBFILEMASK key name:

```
{ IBSTYLE n\ }
{ IBFILEMASK *.dat }
{ IBFILEMASKTYPE 0 }
{ IBFILEMASKCASE 0 }
{ LISTFULDIR 0 }
```

For IBFILEMASK , the file mask string can be any string that excludes spaces. Spaces can cause unexpected errors.

For example, *.dat, test*, test*.dat, or t?st.dat, ^\\d+.dat.

Only "*" and "?" are supported as wildcard .

The file mask string also can be a regular expression such as ^\\d+.dat.

IBFILEMASKTYPE indicates if this mask uses a wildcard or regular expression.

- "0" indicates a wildcard.
- "1" indicates regular expression.

IBFILEMASKCASE indicates if the mask is sensitive to case. This configuration works only for wildcard. Regular expression is always case sensitive.

- "0" indicates not case sensitive,
- "1" indicates case sensitive.

You can use `hcinetcheck` to verify the configuration.

If this item is blank, then all the files in IBDIR are added to inbound file list without restriction.

Usage example

An Integration Engineer has the task of configuring Fileset-local with a file mask.

The Fileset-local protocol is configured to process only files ending with .dat.

A new thread is created and **fileset-local** is selected as the protocol.

The **Properties** button is clicked to display the **Fileset/FTP Local Protocol Properties** dialog box.

From the dialog box, the correct directory is specified and .dat is used as the file mask.

HTTP Client protocol

The HTTP (HyperText Transfer Protocol) Client protocol driver provides an HTTP interface that is highly programmable. HTTP transmits messages to and from remote web servers or other web-enabled applications.

The HTTP Client protocol has a **Use advanced scheduling** option that is used for recurring events.

HTTP client proxy support

HTTP Clients (for example, web browsers) are often required to use what is called a proxy server for requesting documents. When a proxy server is employed, all HTTP requests are sent directly to the proxy server by the client. The proxy server then forwards the requests to the appropriate HTTP server.

In this model, the only server that the HTTP Client connects to directly is the proxy server. This is in contrast to the traditional model, where requests are made to the HTTP servers directly by the client.

Typical applications are:

- To share a single Internet connection. Only the proxy server must have Internet access. All other computers that use the single Internet connection must only be on the LAN. In this way, they can send requests to the proxy server running on the machine with the Internet connection.
- To provide increased security by requiring that only one computer, the one with the proxy server, has access to the Internet through a firewall.
- To locally cache web pages for increased performance.
- To control access to certain web sites. This is content filtering.

To configure the proxy server, you can specify these from the dialog box:

- The host name or IP address of the proxy server.
- The port number on which the proxy server is listening.
- A user name and password to support proxy authentication. This is optional.

Note: Using a proxy server is faster than using direct connections. This is when the proxy server is a caching server with an effective caching mechanism and multiple requests are made for the same document. In all other cases, using a proxy server is considerably slower than using direct connections.

Proxy server configuration

Points to remember when using the proxy server:

- If neither the host name/IP nor port for the proxy is specified, then the proxy feature is inactive. HTTP requests are made directly to the target HTTP servers.
- If a host name is supplied, then that host is used to proxy all requests.
- If no port is supplied, then 8080 on the proxy server machine is used by default.
- If a port is supplied, but not a host, then the proxy server is assumed to be listening at the specified port on *hostname*.

This configuration information is stored in the NetConfig configuration file and specified as arguments to the `httpget`, `httpput`, and `httppost` commands as follows:

```
{PROXY {USER username} {PASS password} {HOST proxyServerMachine} {PORT proxyServerPort} }
```

PROXY is a container key that is a list of proxy configuration options:

- `USER username` is the user name that the connection uses to authenticate the user to the proxy server. If the proxy server is not authenticating, then this field is still sent, but is ignored by the proxy server.
- `PASS password` is the password associated with the user name. If the proxy server is not authenticating, then this field is still sent, but is ignored by the proxy server.
- `HOST proxyServerMachine` is the host name of the computer on which the proxy server is running.

- PORT *proxyServerPort* is the IP port on which the proxy server listens for connections.

HTTP Client protocol: HTTP Options tab

HTTP pane

This table shows the HTTP configuration parameters:

Parameter	Description
URL	Specify the URL with which the engine is to interact. This can be the address of a resource on the web, such as a remote document or CGI script.
Headers	Specify a Tcl keyed list of valid HTTP client headers. This list is made up of name-value pairs, consistent with the HTTP Client protocol. For example: <pre>{Accept text/*} {Cookie id=141} {User-Agent CL6.2}</pre>
Method	Click the arrow to select the HTTP method from the menu: <ul style="list-style-type: none"> • GET is mainly used for retrieving documents, but is also used with scripts if parameters are passed in the URL itself. • POST is used for CGI scripts and forms, passing data within the body of the HTTP transmission. • PUT is used for sending local data to a remote web server, if permissions permit.
Timeout	Specify the time-out value in seconds. This is the time for the libcurl transfer operation to take. If no time-out is defined, then there is no time-out limit for the data transfer.
Local Binding Address	Specify the IP address or a resolvable host name, or click List to make a selection. This field can be left blank in non-load balanced environments.
HTTPS	Select this to enable the security feature, which uses HTTPS for connecting to secure sites. Click ConfigureHTTPS dialog box, where HTTPS settings such as certificates and keys are made. to open the

Parameter	Description
Authorization	HTTP authorization is used when connecting to a web server that does not use HTTPS, but instead uses a user name/password access requirement. Select this to use authorization information such as user name and password. Click Configure to open the Authorization dialog box.
Proxy	Select this to enable the HTTPS Proxy server. Click Configure to open the Proxy Configuration dialog box, where the host, port, user ID, and password are configured.

HTTPS configuration

Select **HTTPS** on the **HTTP Protocol Properties** dialog box and click **Configure**. See [Protocol GUI security settings](#).

This tables shows the HTTPS parameters:

Parameter	Description
Mode	Select the mode from the menu.
SSL Protocol	This is used to select the openssl version from the menu.. When you select an SSL protocol, a description of the selected protocol is shown in the comment field together with the Mode description.
SSL Cipher Suites	Set ciphers in this field. If no cipher is set, then the default cipher suites are used. If this field is not set, then the default cipher suites are used. <ul style="list-style-type: none"> Anonymous mode: server: !DEFAULT:HIG:ADH Client: ALL Non-anonymous mode: HIGH:RC4+RSA:+MD5:!DHE:!3DES:!EXP:!ADH:!AES256-SHA:!AES128-SHA:!EDH:!aNULL:!eNULL:!NULL

Driver Control Procedures pane

This tables shows the driver control procedures parameters:

Parameter	Description
Driver Mode	<p>Select the mode from the menu:</p> <ul style="list-style-type: none"> • Message Driven enables the Query TPS similar to an Outbound Data TPS. Each message arriving on the outbound data queue enables the Query TPS in RUN mode. There is no query interval involved in Message Driven mode. • Time Driven repeatedly enables the Query TPS according to the given query interval. Use this option to perform repeated web queries on a static location over a certain interval. For example, this can be used to periodically query a remote web server for documents that have been modified from the last query, and then retrieve any newly published content. <p>The Query TPS is called with a TPS Mode of RUN in Message Driven mode, and TIME in Time Driven mode.</p> <p>Use advanced scheduling is available only in Time Driven mode.</p>
Query TPS	<p>Click Edit to open the TPS Editor, from which you can specify one or more Tcl procedures to invoke by the HTTP Client driver. You can also configure custom arguments.</p> <p>For a POST, <code>httpQuery</code> is required, as it is the system-provided Tcl procedure to use. When POST is the Method, you must use the <code>{MSGUSE DATA}</code> argument. This tells the <code>httpQuery</code> proc to use the message content as the DATA to be POSTed, instead of the URL to use (the default).</p>
Query Interval	<p>Specify the time interval in which to invoke the TPS specified in the Query TPS option. This option is available only in Time Driven mode.</p>
Use advanced scheduling	<p>Select this to define when a selected TPS should be run, according to an event-based schedule, instead of a query interval. Then, click Setup to open the Scheduling dialog box.</p>

These settings control the interaction between the HTTP Client driver and its custom UPoCs.

Start-Up Procedures pane

Click **Edit** to open the **TPS Editor**. Use this dialog box to select the procedures to run when the thread starts. This is a TPS (Tcl Procedure Stream). Use it to trade messages with the remote end.

Tcl interface

The HTTP Client driver is similar to a UPoC driver. This uses the Network Configurator protocol dialog box to configure and pass values to the Query TPS. All actual HTTP interactions are made through Tcl function calls provided by the system.

The purpose of the NetConfig settings is to provide the defaults for access by the TPS. Usually, when calling a TPS, a list of values is passed in, which are referenced by the Tcl code. Usually, this list consists of four values, but HTTP Client adds a fifth value, `CFGs`, in its Query TPS:

- `CONTEXT`
- `ARGS`
- `MODE`
- `VERSION`
- `CFGs`

For example:

```
httpQuery
{CONTEXT httpc_query}
{ARGS { }}
{MODE start}
{VERSION 3.0}
{CFGs {
  {URL cmhspanther/movies/backup/dir1/}
  {METHOD GET}
  {HEADERS {
    {Connection "Keep Alive"}
    {User-Agent "Cloverleaf Integration Services 6.2"}
    {Accept "image gif"}
  } }
  {AUTH {
    {CREDENTIALS d2VibWFzdGVyOnpycW1hNHY=}
  } }
  {HTTPS {
    {MODE ClientAuth}
    {CA_FILE D:/certs/ca/CaCert.pem}
    {CA_PATH D:/certs/ca/newcerts}
    {CERT_FILE D:/certs/client/ClCert.pem}
    {PRIVATE_KEY D:/certs/client/ClPrivKey.pem}
  } }
} }
```

When you use `CFGs` values, NetConfig values are referenced from within the TPS. You can use the values in `CFGs` as arguments to the HTTP Tcl function calls.

As an added convenience, the `CFGs` list is provided in a format readily usable by the HTTP Tcl functions. They can be passed as arguments to these functions as-is.

Time-out

To configure the time-out, specify the time in seconds for the libcurl transfer operation to take. If no time-out is defined, then there is no time-out limit for the data transfer.

`libcurl` is the name of the underlying library that is used to implement cURL. This is the module that does all the network and protocol work. It uses the options and configuration information from NetConfig to invoke the `libcurl` functions.

In the current TPS, `TIMEOUT` is defined and the default is set to -1. `NTIMEOUT` is set by the dialog box. The engine uses dialog box time-out, set as `NTIMEOUT`, only if `TIMEOUT` is set to -1 on the TPS.

Connection time-out configuration

This configuration is optional. The time-out value is the maximum time in seconds for the connection to the server to take. To configure the connection time-out value, you must change the value of the `CONNTIMEOUT` configuration key by editing the NetConfig file manually.

To configure the connection time-out value find the `CONNTIMEOUT` key:

```
{ PROTOCOL {
{ CA_FILE {} }
{ CA_PATH {} }
{ CERT_FILE {} }
{ CREDENTIALS bnVsbDpudWxs }
{ HEADERS {} }
{ IS_AUTH 0 }
{ IS_HTTPS 0 }
{ IS_PROXY 0 }
{ IS_TIMED_QUERY 0 }
{ LOCAL_IP {} }
{ METHOD GET }
{ MODE {} }
{ PASS {} }
{ PASSWORD {} }
{ PRIVATE_KEY {} }
{ PROXY {{HOST {}}} {PORT {} } {CREDENTIALS {} } {USER {} } {PASS {} } }
{ QUERY_INTERVAL {} }
{ QUERY_TPS {{ARGS {} } {PROCS {} } }
{ NTIMEOUT 10 }
{ CONNTIMEOUT {} }
{ TYPE http-client }
{ URL {} }
{ USER {} }
} }
```

Note: If `CONNTIMEOUT` is blank, then the default value of five minutes is used.

HTTP Client protocol: cURL Options tab

The **cURL Options** tab consists of a table composed of key-value pairs for cURL options.

This table contains the Name, or Key, and Value columns. Every row in the table is a key-value pair for a cURL option.

Supported parameters:

- `long`
- `object pointer` This only includes a `*` character to a zero-terminated string.
- `curl_off_t`

cURL does not support the function pointer parameter and other object pointers, for example, `FILE *`.

If a user-defined cURL option conflicts with one that the system has used, then the user-defined option overwrites the existing one.

If you define the same cURL option twice, then the latter one overwrites the former one.

To help in performing cURL-related testing before configuration, a cURL command-line tool can be found in `$HCIRoot/bin`.

Supported cURL options

See [Fileset FTP protocol: cURL Options tab](#).

Generic Java driver

Note: In the Java and database protocols, one process only supports one java or database protocol. They cannot be in the same process. In the Java driver, you can also run multiple instances in a single process.

The generic Java driver differs from existing system drivers. It requires user additions that provide a framework for Java code. This is used to implement the functionality of a system driver.

This driver differs from the UPOC type driver. It is fully threaded, provides several features not possible in a UPOC driver, and places no restrictions on the use of Java threads in the implementation.

The Java driver also permits multiple instances to run in a single process.

When using the generic Java driver, you can send inbound messages to the system engine through Java code. You can also retrieve outbound messages from the system engine through Java code.

The generic Java driver runs independently from the engine. For example, the engine does not stall when the driver is running. Data is passed between the driver and engine asynchronously. The driver has access to store, retrieve, and remove messages from its own queue in the recovery database.

Server interface

The programmatic interface includes multiple Java methods, most of which are optional. These are configured in the `ini` file. These are described in [JDDK overview](#).

Generic Java Driver type in NetConfig

```
PROTOCOL {
  { IS_TIMED_QUERY {} }
  { QUERY_INTERVAL {} }
  { TYPE java }
```

Selecting the java protocol and then clicking Properties opens the **Java Driver Protocol Properties** dialog box.

If **Use advanced scheduling** is selected, then **Query Interval** is disabled and **Setup** is enabled. Clicking **Setup** opens the **Scheduling** dialog box. For more information on advanced scheduling, see [Network Configurator schedule configuration](#).

Java protocol configuration

For configuration details, see [JDDK overview](#).

Server port configuration

Use the **Server Port** dialog box to get an available port at configuration time and to verify whether the selected port is currently in active use.

On the IDE, server ports are configured at:

- **Network Configurator Properties tab > Inter-Site Routing Port**
- **TCP/IP Protocol Properties dialog box > Server/Multi-Server Type > Port**
- **PDL TCP/IP Protocol Properties dialog box > Server/Multi-Server Type > Port**

The behavior for configuring the server port is the same for all of these locations.

Select button and Select Port dialog box

On these dialog boxes, **Select** or **List** is next to **Port**.

Clicking **Select** or **List** opens the **Select Port** or **Ports List** dialog box. The available dialog box depends on the selected protocol.

Ports List dialog box

The Ports List contains well-known services. Double-clicking a service places the selected item into the **Port** field.

Select Port/Find Port dialog boxes

On protocols with **Select**, clicking **Verify** calls the command line to check whether the value of the **Port** field is an unused port number.

- If the port is available, then the **Port Verification** dialog box opens, stating it is available.
- If the port is unavailable, then the **Port Verification** dialog box opens, stating it is not available.
- Port numbers that are specified outside the range of 0 to 65535 open an "invalid port number" message. This happens when you click **OK/Verify** on the **Select Port** dialog box.
- Otherwise, clicking **OK** populates the port value.

For information on command line usage, see [hcggetnextport](#) on page 1347.

Clicking **Find** on the **Select Port** dialog box opens the **Find Port** dialog box.

Specify the **Search Scope** by selecting an option and then clicking **OK**. This calls the `hcggetnextport` command line. The default search scope is **Search All Sites**.

When an option is selected, the command line attempts to find an un-configured and inactive port using the specified **Port Range** and **Search Scope** parameters.

When a port is found, you are prompted to confirm the found port.

- **Yes** closes the prompt and the **Find Port** dialog box. It then returns to the **Select Port** dialog box with the **Port** field showing the found port.
- **No** returns to the **Find Port** dialog box.

When **aPort Range** and **Search Scope** are selected, but no available port is found, a prompt opens asking for confirmation to modify the search parameters. Clicking **Yes** returns to the **Find Port** dialog box.

LU 3 protocol

The LU 3 protocol is used by application programs that communicate with a printer through IBM 3270 data streams. The system hub receives the data streams.

LU 3 Options pane

For **Profile**, specify the name of the site information profile for this connection. Click **List** to view a predefined list of profiles.

LU 3 Auto-Reconnect Options pane

This table shows the auto-reconnect options:

Option	Description
Auto-Reconnect	Click to automatically reestablish a broken connection.
Reopen Time	Specify the minimum number of seconds to wait after connection failure before reconnection. The default is 5.

LU 3 Start-Up Procedures pane

Click **Edit** to open the **TPS Editor**. Use this to select the Tcl procedures to run when the thread starts. This is a TPS (Tcl Procedure Stream). Use it to trade messages with the remote end.

LU 6.2 APPC protocol

The LU 6.2 APPC protocol driver manages an APPC (Advanced Program-to-Program Communication) with another entity using LU 6.2. The driver can be configured to await (Listen) or initiate (Call) connections.

LU 6.2 is an asynchronous peer-to-peer SNA (Systems Network Architecture) Server LU-type protocol. SNA Server is an IBM product that is a set of programs and data files providing transparent access to SNA network resources. SNA can only be configured on AIX.

Knowledge of SNA products is required for LU 6.2 configuration.

SNA consists of these main levels:

- Physical Unit (PU).

This level represents a distributed control element in a network that manages and controls resources directly attached to it. For example, Token Ring, TCP/IP, SDLC, and so on.

The PU is configured at both ends, AIX and mainframe, with the type of network, network name, and other control data.

The PU must establish communications before anything can happen over the network.

This is also referred to as the Link Level.

- Logical Unit (LU).

This level sets up communications between local programs and other networks. In SNA, there are many LUs, each with a specific function, such as Terminal Emulation (LU 2), Printers (LU 3), and others.

LU6.2 is known as APPC. Within APPC, there are two types of LUs:

- Dependent:

These LUs are subservient to the mainframe and can support only a single program-to-program conversation.

- Independent:

These LUs are controlled also by the mainframe but can support multiple program-to-program conversations over a single LU.

A Local LU is defined on the AIX side. This must be a name that is known to the mainframe. The mainframe defines the LU (APPLID), which becomes known as the Remote LU on the AIX side. The two sides then agree upon a mode name. The mode is a table of characteristics for each side.

When a logical session is attempted, the mainframe acquires the LU on the AIX side. A Bind message is issued and the negotiation begins. If the negotiation is successful, then the LU-to-LU session is implemented. Only then can program-to-program communication begin.

- Conversation.

In this level, one program is designated the Caller and another as Listener.

- The Caller issues an Allocate request.

- The other side is listening, and runs its program and responds positively to the Allocate. This is where program-to-program communications begin.

LU6.2 is a half-duplex protocol. After the Caller sends, it can reverse the sense of a logical switch to permit the other side to send. Then, the sense is reversed again, and this continues.

Nothing can happen at any level until the previous level is successful.

The system is involved only at the Conversation level.

Communication methods

After SNA is configured properly and sessions are available, there are two major methods of communicating:

- Low-level APPC invocations.
- High-level Application Program Interface (API), known as Common Programming Interface for Communications (CPIC).

The AIX version uses CPIC invocations to implement the APPC communications interface. CPIC is a well-documented API within IBM, using function invocations such as Allocate, Send_Data, Receive, Deallocate, and so on.

SNA server

Application programs use an SNA Server to communicate with traditional 3270, Remote Job Entry (RJE), and peer applications within an SNA network. SNA formally defines the functional responsibilities for components of a data communications system. It also specifies how those components must interact.

SNA Server provides an application programming interface to SNA Logical Unit (LU) 0,1, 2, 3, and 6.2 protocols over a Physical Unit (PU) Type 2.1.

SNA Server requires one or more of these adapters on the system:

- Token Ring High-Performance Network Adapter, with appropriate cables to attach to a Token Ring LAN.
- ETHERNET High-Performance LAN Adapter, with appropriate cables to attach to an Ethernet or IEEE 802.3 LAN.
- X.25 Interface Co-Processor/2, with appropriate cables to attach to an X.25 packet switching network.
-

Configuration in UNIX

The driver can be configured and commanded to issue and handle some LU 6.2 control modes and verbs.

Note: On AIX systems, the supported SNA product is IBM Communications Server.

Name pane

This table shows the parameters for the **Name** page:

Symbolic Name	Specify the CPIC Symbolic Name. This is defined when configuring IBM Communication Server SNA (AIX).
Connection Type	Select the appropriate connection type: <ul style="list-style-type: none"> • Call, or initiate connection, contacts a remote entity and requests a connection with an RTPN (Remote Transaction Program Name). • Listen, or await connection, connects user-specified local TPN (Transaction Program Name) requests to the driver for registering with SNA. • If the thread is not running, then applications attempting to connect to its TPN receive "TPN Not Recognized" errors.

Call Properties pane

This table shows the parameters for the **Call Properties** page:

RTPN	For a Call connection, specify the RTPN (Remote Transaction Program Name). RTPN is an application that uses this connection to send data to another application.
Delay	<p>Select this to delay connection allocation until there is a message to write. When you use this option, the driver allocates a conversation only when required for an outbound message.</p> <p>When this option is not selected, the driver attempts to allocate a conversation with the LISTENER as soon as possible. That is, during thread startup, after message writes if Deallocate is set, and so on.</p>

Security pane

This table shows the parameters for the **Security** page:

Secure	Select this to configure the driver to use the LU 6.2 security options. If this option is used, then the driver requests SECUR_PGM and includes the configured User ID and Password values. Otherwise, it uses SECUR_NONE. This option is available only if Call is selected.
User ID	Specify the user ID. This option is available only if Secure is selected.
Password	Specify the log-in password. This option is available only if Secure is selected.

Listen Properties pane

For a Listen connection, specify the **TPN** (Transaction Program Name). TPN is an application that uses this connection to receive data from another application.

Sync Level pane

This table shows the parameters for the **Sync Level** page:

Conversation sync level	<p>None means SNA provides no synchronization services.</p> <p>Confirm prepares the conversation for possible CONFIRM/CONFIRMED verb use. If this option is used, then select Auto-confirm to automatically send a CONFIRM request with every message.</p>
-------------------------	--

Auto-confirm

Select this to request a Confirm for every message that is written to the connection. This option is available only if **Confirm** is selected.

Miscellaneous Properties pane

This table shows the parameters for the **Miscellaneous Properties** page:

Control Messages

Select this to enable the driver to use the DRIVERCTL metadata. Modify the driver's behavior by embedding certain control information in the metadata field. The driver also reports the state of the connection in the inbound message's DRIVERCTL field values. Setting this option permits greater flexibility and control, but at the expense of memory and speed.

Deallocate

Select this to configure the driver to deallocate the connection, or conversation, after every message send.

EBCDIC Alloc

Select this to convert the RTPN/TPN to EBCDIC format. If this option is not selected, then ASCII is used.

Flush

Select this to automatically flush the buffer after each operation.

Auto-Reconnect

Select this to automatically reestablish a broken connection.

Full-Duplex

Select this to place the conversation in a RECEIVE state after each message is sent. This requires a response for each request. The remote application must explicitly pass SEND control back to the driver. This option is unavailable if the connection is send-only.

Reopen Time

Specify the minimum number of seconds to wait after connection failure before reconnection. The default is 5. This option is available only if **Auto-Reconnect** is selected.

Start-up Procedures pane

Click **Edit** to open the **TPS Editor**. Use this to select the Tcl procedures to run when the thread starts. This is a TPS (Tcl Procedure Stream). Use it to trade messages with the remote end.

Initialization

A Call thread allocates a conversation to initiate the connection, at start-up time or outbound message write time if **Delay** is selected.

If **EBCDIC Alloc** is selected, then all strings that are copied into the allocation request are transliterated into EBCDIC. Otherwise, the driver uses ASCII.

The configuration value determines the conversation sync level. The driver always requests a MAPPED conversation.

If the security option is used, then the driver requests SECUR_PGM and includes the configured **User ID** and **Password** values; otherwise, it uses SECUR_NONE.

After the conversation is established, the driver sends a FLUSH. Because the CALL establishes the conversation, it has initial control of the conversation and is in the SEND state.

The Listen thread begins by opening the `/dev/sna/profile_name` pseudo-device where *profile_name* is the name of the configured SIP.

With that pseudo-device, it registers itself with SNA as the configured TPN.

If **EBCDIC Alloc** is selected, then the driver registers this string using EBCDIC instead of ASCII.

When the TPN is registered, the thread awaits conversation allocation requests. When one arrives, the driver retrieves the parameters in the request and checks the requested TPN and profile names. If all is well, then the driver allocates a conversation, enters the RECEIVE state, and waits for message data.

Inbound messages

When the driver is in the RECEIVE state, the remote application can send data at any time. As the driver receives data packets, it appends them to a message object.

The driver recognizes these flags:

- SEND_RECEIVED
The sender grants control to the driver; the driver moves into the SEND state.
- CONFIRM_RECEIVED
The driver responds with CONFIRMED.
- CONFIRM_SEND_RECEIVED
SEND_RECEIVED and CONFIRM_RECEIVED
- NORMAL_DEALLOCATE_RECEIVED
Deallocates the conversation per peer's request.
- CONFIRM_DEALLOCATE_RECEIVED
CONFIRM_RECEIVED and NORMAL_DEALLOCATE_RECEIVED

Note: The CONFIRM flags display only when you use Sync Confirm.

SNA control flags, attached to a packet, identify the packet as the last in a chain. In this case, the driver passes the message into the engine for processing.

Each packet may also have other SNA control flags attached to it. If the packet contains a `DEALLOCATE` flag, then the driver stops reading; otherwise, it collects the flag and acts on it after the message read is complete.

If the control flag is set, then the driver includes a description of the connection state in the message's `DRIVERCTL` string.

The driver creates a keyed list with these keys:

- `DRIVER appc`
Identifies the driver
- `STATE send or receive`
Driver state after reading the message
- `CONTROL confirm, dealloc, send`
Control flags received with the message

Some examples are:

- Data arrived with `CONFIRM_SEND_RECEIVED`:

```
{DRIVER appc} {STATE send} {CONTROL {confirm send}}
```

- Data arrived with `CONFIRM_RECEIVED`:

```
{DRIVER appc} {STATE receive} {CONTROL {confirm}}
```

- Data arrived with `NORMAL_DEALLOCATE_RECEIVED`:

```
{DRIVER appc} {STATE receive} {CONTROL {dealloc}}
```

Unless the peer gives control of the conversation to the driver and moves it into the `SEND` state, the driver remains in the `RECEIVE` state.

If a read error happens when receiving message data, then the driver deallocates the conversation. Then, it closes the connection, discards the unfinished data, and enters the `ERROR` state.

Outbound messages

When writing a message, the driver sets the control flags according to the default configuration values.

The `CONTROL` flag is used through the **Control Messages** option. The driver examines the message's `DRIVERCTL` string for an `APPC` key. That is, there might be control strings for multiple drivers.

If the driver finds one, then it parses the string as a keyed list with boolean values and recognizes these keys:

- `CONFIRM`
- `DEALLOCATE`
- `FLUSH`
- `PREPARE_TO_RECEIVE`

Some examples are:

- Request confirmation and put the peer into SEND state:

```
{APPC {{CONFIRM 1} {PREPARE_TO_RECEIVE 1}}}
```

- Flush the data stream and request a deallocation:

```
{APPC {{CONFIRM 0} {DEALLOCATE 1} {FLUSH 1} {PREPARE_TO_RECEIVE 0}}}
```

If the DRIVERCTL string requests a deallocation, then it is handled after the write; otherwise, the default control flags are handled as follows:

```
if (Full-Duplex)
    send PREPARE_TO_RECEIVE verb
else if (Deallocate after every write)
    deallocate conversation
else if (Flush after every write)
    send FLUSH verb
endif
```

The Full-Duplex flag is used through the **Full Duplex** option. This is set, or the DRIVERCTL string is set, to the PREPARE_TO_RECEIVE key. The driver retains control of the conversation and remains in the SEND state.

WebSphere MQ protocol

WebSphere MQ enables application programs to communicate with each other using messages and queues (asynchronous messaging). This provides assured, once-only delivery of messages. The application programs can be separated. The program sending a message can continue processing without having to wait for a reply from the receiver. If the receiver is temporarily unavailable, then the message can be forwarded at a later time.

Note: The WebSphere MQS protocol driver requires installation of the IBM WebSphere MQ software. This lets the engine interface with IBM's WebSphere MQ software.

Before applications can send any messages, a Queue Manager and queues must be created. Queues are managed by the Queue Manager. The Queue Manager provides messaging services for the applications and processes the MQI (Message Queue Interface) invocations they issue. The MQI is a set of invocations that applications use to ask for the services of a Queue Manager. The Queue Manager ensures that messages are put on the correct queue or that they are routed to another Queue Manager.

Configure the Websphere MQ queues using tools provided by IBM.

Use Network Configurator (hcinetconfig) to name the Queue Manager, inbound queue, and outbound queue.

The Queue Manager may be on the local machine if using Websphere MQ, or on another machine if MQClient is installed on the local machine. This Queue Manager must be previously created and started. The command server must also be running for that Queue Manager.

This table shows the guidelines for the WebSphere MQ protocol:

Guideline	Description
Queue Manager	You must name the Queue Manager using IBM naming conventions, with a maximum of 48 characters.
Queue	You must name the queues using IBM naming conventions, with a maximum of 48 characters.
Channel	<p>A channel is used for accessing Queue Managers on other machines through MQClient.</p> <p>You must:</p> <ul style="list-style-type: none"> • Name the channel using IBM naming conventions. • Define it in the Queue Manager on the server. • Set the channel type to SVRCONN. • Set the MCAUSER to <code>user@domain</code>.

If required, then you can specify that a local queue is opened for both Getting and Putting.

Remote queues are opened for Putting only.

The Queue Manager returns an object handle if the open request is successful. The application specifies this handle, together with the connection handle, when it issues a PUT or a GET call. This ensures that the request is carried out on the correct queue.

You can change certain message descriptor fields using Network Configurator and Tcl procedures.

See [Websphere MQ configuration details](#).

Queue Manager pane

This table shows the parameters for the Queue Manager name:

Parameter	Description
Queue Manager Name	Specify the Queue Manager name to which the system connects.

Parameter	Description
Commit Interval	<p>Specify the number of messages to be read and processed before the Queue Manager commits them from the inbound queue. The minimum is 1.</p> <p>Fewer calls are made to process messages when Maximum Messages Per Read (on the MQ Inqueue Advanced Properties dialog box) and Commit Interval have a value greater than 1.</p> <p>If anything happens to the thread between the time when a message is read and when it is committed, then the message is redelivered. It is redelivered to when the thread comes back up.</p> <p>If Commit Interval is greater than the number of messages read, then this could result in messages not being committed for a longer time period.</p> <p>Note: Commit Interval can increase performance, but should be used with care in cases where redelivery of a message is unacceptable.</p>
Read Interval	<p>Specify the length of time, in seconds, to wait before reading a message from the inbound queue or from the reply queue.</p>

Inbound pane

This table shows the parameters for the Inbound pane:

Parameter	Description
Enable inbound queue	Select this to open and retrieve messages from the queue. Otherwise, the inbound queue is ignored.
Queue Name	Specify the inbound queue name from which messages are opened and retrieved.
Advanced	<p>Click to open the MQ Inqueue Advanced Properties dialog box.</p> <p>Note: This option is available only when Enable inbound queue is selected.</p> <p>See MQ Inqueue Advanced Properties dialog box.</p>

Outbound pane

This table shows the parameters for the Outbound pane:

Parameter	Description
Enable outbound queue	Select this to open and place messages on the queue. Otherwise, the outbound queue is ignored.

Parameter	Description
Queue Name	Specify the outbound queue name from which messages are opened and placed.
Advanced	<p>Click to open the MQ Outqueue Advanced Properties dialog box.</p> <p>Note: This option is available only when Enable outbound queue is selected.</p> <p>See MQ Outqueue Advanced Properties dialog box.</p>

Client Information pane

This table shows the parameters for the Client Information pane:

Parameter	Description
Server Name	<p>Specify the server name containing the MQS server. UNIX server names are case-sensitive.</p> <p>If this is blank, then the thread works as a server.</p> <p>If this is filled with the host name to which this thread is to connect, then the thread works as a client.</p>
Channel Name	Specify the channel name that was previously created on the MQS server.
Admin Queue Name	Specify the queue name. If left blank, then the engine uses the default value (SYSTEM.ADMIN.COMMAND.QUEUE).
User ID/Password	
Port Number	<p>Specify the port number to use when connecting to the MQS server. This field is optional and defaults to 1414 if left blank. If a port number other than 1414 is specified, then a listener must be started on the server using this port number as an argument. Click List to open the Ports List dialog box to browse an alias list, where you can assign a port with an alias.</p>
Time Out	<p>Specify a time out value, in the range of 1 to 60. If left blank, then the engine uses the default value (1).</p> <p>Note: This information applies only to platforms supporting the MQS Client.</p>
SSL	Click Configure to open the MQS SSL/TLS Properties dialog box. See MQS SSL/TLS Properties dialog box .

Start-up Procedures pane

Click **Edit** to open the **TPS Editor**. Use this to select the Tcl procedures to run when the thread starts. This is a TPS (Tcl Procedure Stream). Use it to trade messages with the remote end.

MQ Inqueue Advanced Properties dialog box

This table shows the parameters on the **MQ Inqueue Advanced Properties** dialog box:

Option	Description
Open Options	Click List to select the Out and In values of messages to be retrieved. Input and Output options are mutually exclusive.
Close Options	Select from the menu which values to use upon close of file.
Get Msg Options	Click List to select the Out and In values of the Get message.
Put Msg Options	Click List to select the Out and In values of the Put message.
Wait Interval	Specify the interval in milliseconds that the invocation of MQGET waits for a suitable message to arrive. This value is ignored if Get Msg Options is MQGMO_WAIT, or MQGMO_SET_SIGNAL is not specified.
Max Messages Per Read	Specify the number of messages to be read and processed at each read interval.
Use MQ Buffer	Select this to use the MQ buffer. If selected, then incoming messages are first placed into a fixed buffer before being copied to the message.
Copy MQMD	Select this to copy MQMD (MQ Message Descriptor) Properties. If selected, then MQMD values of incoming messages are copied to DRIVERCTL.

MQ Outqueue Advanced Properties dialog box

This table shows the parameters on the **MQ Inqueue Advanced Properties** dialog box:

Option	Description
Queue Type	Select the type of queue from the menu: Local or Remote .

Option	Description
Open Options	Click List to select the Out and In values of messages to be retrieved. Input and Output options are mutually exclusive.
Close Options	Select from the menu which values to use upon close of file.
Put Msg Options	Click List to select the Out and In values of the Put message.
Get Msg Options	Click List to select the Out and In values of the Get message.
Reply To Model Queue	Specify the name of the reply-to model queue to use as a template when creating permanent dynamic queues for replies. The DEFTYPE attribute of this model queue must be set to PERMDYN.
Query Depth Each Message	Select this to query the message depth.
MQMD Features	Click to open the MQ Message Descriptor Properties dialog box. Use this dialog box to configure MQMD (MQ Message Descriptor) features. See Websphere MQ configuration details .

Websphere MQ configuration details

For message header access, Websphere MQ adds a message descriptor, MQXQH structure. This contains control information for each message before placing the message into a transmission queue. The MQXQH structure has five fields, the last of which is MsgDesc. The MsgDesc parameter is another structure (MQMD). It is the MQMD, MQ Message Descriptor, information that remains with the message after it arrives at its destination queue.

Note: To preserve the MQMD values, the COPY MQMD switch must be set for the inbound queue. Do this by selecting **Copy MQMD** on the **MQ Inqueue Advanced Properties** dialog box.

The MQMD structure has fields which are accessible to users. Ten of these fields are most often user-modified:

- MSGTYPE: Message type.
- EXPIRY: Message lifetime.
- FORMAT: Format name.
- PRIORITY: Message priority.
- PERSISTENCE: Message persistence.
- MSGID: Message identifier.
- CORRELID: Correlation identifier.
- REPLYTOQ: Name of reply queue.
- REPLYTOQMGR: Name of reply Queue Manager.
- REPORT: Options for report messages.

The message and correlation identifiers are particularly important for matching messages with replies, which are found in physically different locations. Messages are generally put into a remote queue for sending to a remote server. The reply from the remote server is found in a separate local queue established to receive replies only.

The remainder of the MQMD fields are:

- **STRUCID:** Structure identifier.
- **VERSION:** Structure version number.
- **FEEDBACK:** Feedback or reason code.
- **ENCODING:** Data encoding.
- **CODEDCHARSETID:** Coded character set identifier.
- **BACKOUTCOUNT:** Backout counter.
- **USERIDENTIFIER:** User identifier.
- **ACCOUNTINGTOKEN:** Accounting token.
- **APPLIDENTITYDATA:** Application data relating to identity.
- **PUTAPPLTYPE:** Type of application that Put the message.
- **PUTAPPLNAME:** Name of application that Put the message.
- **PUTDATE:** Date when message was Put.
- **PUTTIME:** Time when message was Put.
- **APPLORIGINDATA:** Application data relating to origin.
- **GROUPID:** Group identifier.
- **MSGSEQNUMBER:** Sequence number of logical message within group.
- **OFFSET:** Offset of data in physical message from start of logical message.
- **MSGFLAGS:** Message flags.
- **ORIGINALLENGTH:** Length of original message.

Storing messages in the recovery database

To minimize any possibility of losing messages before they are stored in the recovery database, set the MQG MO_SYNCPOINT **Get Msg** option. Set this to tell the Queue Manager to lock the messages that are read. Then, set it to wait to delete these messages until the Commit, MQCMIT, function is called. The Commit function is not called until the number of messages, specified under COMMITINTERVAL for the Queue Manager, are put into the recovery database.

There is a possibility of the system failing after messages are committed to the recovery database and before the WebSphere MQ Commit function is called. In this case, when operation is resumed, some of the messages in the WebSphere MQ inbound queue are duplicates. They are duplicates of ones that were already stored in the recovery database. If the message IDs, not necessarily Websphere MQ message IDs, are checked before storage in the recovery database, then duplication is avoided. Message ID checking can be performed within a Tcl procedure.

See [Open, Close, Get Message and Put Message options](#).

Matching messages with replies

If the engine is configured to await replies, then there are four ways to match an outgoing message with its incoming reply. The outgoing message is written to the outbound queue. The reply is written to the reply-to queue as specified under `MQMD` in the Network Configurator.

Matching methods depend upon the Report member of `MQMD`:

- If Report equals `MQRO_NONE`, then the first message found in the reply-to queue is processed as the reply.
- If Report equals `MQRO_PASS_MSG_ID`, then the `MsgID` field of the outgoing message is compared to the `MsgID` of the reply.
If they match, then the reply is processed.
- If Report equals `MQRO_PASS_CORREL_ID`, then the `CorrelID` of the outbound message is compared to the `CorrelID` of the reply.
- If Report is set equal to `MQRO_COPY_MSG_ID_TO_CORREL_ID`, then the reply is processed if its `CorrelID` matches the `MsgID` of the outbound message.

For reply queues, you must specify one of these `MQRO` report options:

- `MQRO_PASS_MESSAGE_ID`
- `MQRO_PASS_CORRELATION_ID`
- `MQRO_COPY_MESSAGE_ID_TO_CORRELATION_ID`

This controls how inbound reply messages are matched up with outbound messages. One and only one of these three options can be specified. Other `MQRO` options can be included.

The name of a reply-to queue must be specified in `MQMD` if the engine is configured to await replies. This queue is not necessary, but must be named.

- If the queue does exist when the system is run, then this queue is used for replies.
- If the queue does not exist, then a `Cloverleaf.MODELQ` model queue is created and used as a template. This happens if a model queue is not specified under the outbound queue parameters. This template create a permanent dynamic queue to handle replies.
- Sometimes, the reply-to queue does not exist, and a model queue is specified under the outbound queue parameters. When this happens, the attributes of this model queue are used to create a dynamic queue to handle replies.

Message identifier options

Specify one of these options to control how the **Message ID** of the report message, or reply message, is to be set:

- `MQRO_NEW_MSG_ID`: New message identifier.
This is the default action. It indicates that if a report or reply is generated because of this message, a new **Message ID** is generated for the report or reply message.
- `MQRO_PASS_MSG_ID`: Pass message identifier.
If a report or reply is generated because of this message, then the **Message ID** of this message is copied to the **Message ID** of the report or reply message.
- If this option is not specified, then `MQRO_NEW_MSG_ID` is assumed.

When the Queue Manager generates a report message, it sets the **Message ID** field. This field is set in the way that is specified by the **Report** field of the original message. It uses MQRO_NEW_MSG_ID or MQRO_PASS_MSG_ID.

Message metadata changes and queries

Use `msgmetaget` to query message metadata, and `msgmetaset` to alter message metadata. The arguments for these functions are:

```
msgmetaset msgId key value ?key2 value2 ...?
msgmetaget ?-rw | -ro | -all? msgId ?key?
```

- key in both is DRIVERCTL.
- Items between question marks are optional.
- `-rw` means read-write keys.
- `-ro` means read-only keys.
- `-all` means all keys (default).

For example, if a user must set the **Report** options value in MQMD, any, all, or none of these report types can be requested:

- Exception, which contains these choices, only one of which can be made:
 - MQRO_EXCEPTION
 - MQRO_EXCEPTION_WITH_DATA
 - MQRO_EXCEPTION_WITH_FULL_DATA
- Expiration, which contains these choices:
 - MQRO_EXPIRATION
 - MQRO_EXPIRATION_WITH_DATA
 - MQRO_EXPIRATION_WITH_FULL_DATA
- Confirm On Arrival (COA), which contains these choices:
 - MQRO_COA
 - MQRO_COA_WITH_DATA
 - MQRO_COA_WITH_FULL_DATA
- Confirm On Delivery (COD), which contains these choices:
 - MQRO_COD
 - MQRO_COD_WITH_DATA
 - MQRO_COD_WITH_FULL_DATA
- Positive Action Notification (PAN)
 - MQRO_PAN
- Negative Action Notification (NAN)
 - MQRO_NAN

Note: The PAN and NAN choices (MQRO_PAN and MQRO_NAN) should be mutually exclusive.

To show the above, a `msgmetaset` command to request an `EXCEPTION` report with data and a positive action notification is:

```
msgmetaset msgId DRIVERCTL {MQS {REPORT {MQRO_EXCEPTION_WITH_DATA MQRO_PAN}}}
```

The engine parses the contents of the `DRIVERCTL` string in the message metadata and sets the appropriate options in `MQMD` before sending the message.

Message metadata driver control

The message contains metadata structure. One of the elements of this structure is a character string known as "driver control," referred to as `DRIVERCTL`. This string can be used to insert protocol-specific information. For the WebSphere MQ protocol, this information consists of the `MQMD` values.

You can set any `MQMD` fields that are specified in this keyed list. Similarly, modify this keyed list to contain the `MQMD` values each time a message is read. This facilitates the getting of `MQMD` message values by Tcl. `msgmetaset` (a Tcl extension) is the command used to change the `msg_driver_control` string. The `DRIVERCTL` key value is used with this command to indicate that `msg_driver_control` is to be changed.

Changing MQMD field values

There are three methods available for changing or viewing the `MQMD` field values:

- Method #1: Select values within the Network Configurator.

Indicate the value of these fields one time in the Network Configurator when establishing the WebSphere MQ protocol thread. This is particularly useful when setting the `MQMD` fields. The values are the same for a large number of records to be output from the system.

MQMD Features on the **MQ Outqueue Advanced Properties** dialog box opens the **MQ Message Descriptor Properties** dialog box.

Use this dialog box to specify the values. These values are cached and are not changed unless a Tcl procedure indicates otherwise.

- Method #2: Change or display using Tcl.

Dynamically change the contents of the message metadata string `DRIVERCTL` as each message is put into the outbound queue. You can also display the contents of `DRIVERCTL` as a message is read from the inbound queue. This is only if `CopyMQMD` is set under the inbound queue parameters. This is performed by following the instructions in a Tcl procedure, or Tcl procedure stack.

This Tcl procedure can be associated with any UPoC, including the UPoC protocol. The name of this procedure, along with any arguments, is entered into the NetConfig file through the Network Configurator. To do this:

- 1 On the Network Configurator, click a connection icon and select the **Inbound** or **Outbound** tab.
- 2 Select the appropriate entry and click **Edit** to open the **TPS Editor**.
- 3 Then, click **Add** to select the procedures.

- Method #3: Pass through `MQMD` values.

This method assumes that the messages initially received have originated from a server using Websphere MQ. In this case, the user may keep all, or some, of the `MQMD` parameter values when the message is sent

to another Websphere MQ server. This choice is indicated through the `CopyMQMD` parameter in the inbound queue section. This section is accessed by clicking **Advanced** which opens the **MQ Inqueue Advanced Properties** dialog box.

Selecting `CopyMQMD` for the inbound queue means that the incoming MQMD values are to be copied to the `DRIVERCTL`.

In general, the incoming message contains all MQMD fields. Therefore, selecting `CopyMQMD` for the inbound queue overrides all values that are specified in the `NetConfig` file and pass MQMD values from input to output. This is only if a Tcl procedure is not invoked in-between to change MQMD values.

Open, Close, Get Msg, and Put Msg options

Descriptions of each of these options and their inter-dependencies are provided in the WebSphere MQ documentation.

The **Open**, **Close**, **Get Msg** and **Put Msg** options are selected in Network Configurator from the **MQS Protocol Properties** dialog box when the protocol is `mqs`. Specified **Open** options are used when the WebSphere MQ thread is started. The intention is that the queues are opened only after when the thread is started and closed only after when the thread is stopped.

Open options

- `MQOO_INPUT_AS_Q_DEF`
- `MQOO_INPUT_SHARED`
- `MQOO_INPUT_EXCLUSIVE`
- `MQOO_BROWSE`
- `MQOO_OUTPUT`
- `MQOO_INQUIRE`
- `MQOO_SET`
- `MQOO_BIND_ON_OPEN`
- `MQOO_BIND_NOT_FIXED`
- `MQOO_BIND_AS_Q_DEF`
- `MQOO_SAVE_ALL_CONTEXT`
- `MQOO_PASS_IDENTITY_CONTEXT`
- `MQOO_PASS_ALL_CONTEXT`
- `MQOO_SET_IDENTITY_CONTEXT`
- `MQOO_SET_ALL_CONTEXT`
- `MQOO_ALTERNATE_USER_AUTHORITY`
- `MQOO_FAIL_IF QUIESCING`
- `MQOO_RESOLVE_NAMES`

Close options

- `MQCO_NONE`
- `MQCO_DELETE`
- `MQCO_DELETE_PURGE`

Get Message options

- MQGMO_WAIT
- MQGMO_NO_WAIT
- MQGMO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_NO_SYNCPOINT
- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_NEXT
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_MSG_UNDER_CURSOR
- MQGMO_LOCK
- MQGMO_UNLOCK
- MQGMO_ACCEPT_TRUNCATED_MSG
- MQGMO_SET_SIGNAL
- MQGMO_FAIL_IF QUIESCING
- MQGMO_CONVERT
- MQGMO_LOGICAL_ORDER
- MQGMO_COMPLETE_MSG
- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE
- MQGMO_NONE

Put Message options

- MQPMO_SYNCPOINT
- MQPMO_NO_SYNCPOINT
- MQPMO_NEW_MSG_ID
- MQPMO_NEW_CORREL_ID
- MQPMO_LOGICAL_ORDER
- MQPMO_NO_CONTEXT
- MQPMO_DEFAULT_CONTEXT
- MQPMO_PASS_IDENTITY_CONTEXT
- MQPMO_PASS_ALL_CONTEXT
- MQPMO_SET_IDENTITY_CONTEXT
- MQPMO_SET_ALL_CONTEXT
- MQPMO_ALTERNATE_USER_AUTHORITY
- MQPMO_FAIL_IF QUIESCING
- MQPMO_NONE

Option restrictions

There are certain restrictions for setting options. For example, there are three input choices for **Open** options. These options are mutually exclusive. In addition, none of the input **Open** options can be selected if MQOO_OUTPUT is chosen. Similarly, only one **Close** option can be selected.

Typical selections for the inbound queue are:

- For **Open** options:
MQOO_INPUT_AS_Q_DEF
MQOO_INQUIRE
MQOO_FAIL_IF QUIESCING
- For **Close** options:
MQCO_NONE
- For **Get Msg** options:
MQGMO_WAIT
MQGMO_SYNCPOINT
MQGMO_FAIL_IF QUIESCING
MQGMO_CONVERT
- For **Put Msg** options:
MQPMO_NONE

Typical selections for the outbound queue would be:

- For **Open** options:
MQOO_OUTPUT
MQOO_FAIL_IF QUIESCING
- For **Close** options:
MQCO_NONE
- For **Put Msg** options:
MQPMO_FAIL_IF QUIESCING

If the outbound queue is a local queue, then the **Get Msg** options can also be set.

MQS SSL/TLS Properties dialog box

In the MQS Client Information panel of the **MQS Protocol Properties** dialog box, clicking **SSL** enables **Configure**.

Click **Configure** to open the **MQS SSL/TLS Properties** dialog box.

This table lists the options for MQS protocol SSL/TLS:

Field	Description
Key Repository Stem	<p>The stem name of the key repository file. This is the full path to the file without the <code>.kdb</code> suffix.</p> <p>For example:</p> <ul style="list-style-type: none"> <code>/home/user/client</code> <code>C:\User\client</code> <p>The file name is <code>client.kdb</code>.</p>
Certificate Label	<p>The certificate label to use for the secure connection. This must be lowercase.</p>
Certificate Validation Policy	<p>A check box to specify the certificate validation policy to use. This must be one of:</p> <ul style="list-style-type: none"> <code>ANY MQ_CERT_VAL_POLICY_ANY</code> <code>RFC5280 MQ_CERT_VAL_POLICY_RFC5280</code>
Online Certificate Status Protocol	<p>An edit box to determine which Online Certificate Status Protocol (OCSP) responder to use.</p>
Cipher Specification	<p>A list box for choosing the cipher specification.</p>

PDL drivers

PDL (Programmable Driver Language) drivers manage byte-stream oriented protocols and require PDL/Tcl code to parse incoming data and prepare outgoing data for transmission.

PDL supports asynchronous (PDL Async) and TCP/IP terminal connections. Each must have unique configuration information. After a PDL connection is established, PDLs act the same. Therefore, use the same PDL/Tcl code to manage different connections, independent of the medium.

PDL driver configuration

All PDL drivers are configured with the name of the PDL used to control the connection.

PDL drivers enable the customization of the behavior of a communications interface using the named base protocol. With it, you can add functions, such as defining start and stop characters to bound messages. You can also calculate and add a checksum to a message.

PDL Async protocol

PDL Async is a PDL-based driver for asynchronous serial interfaces. Communications through RS-232 connections are asynchronous in nature. For asynchronous communications, specify the port number and the behavior characteristics such as baud rate, number of data bits, parity, and other characteristics.

PDL Options pane

This table shows the parameters in the PDL Options and PDL Async panes:

Parameter	Description
PDL	Click List to view a predefined list of PDL script names.
Line	The first device name. For example, tty0.
Modem	Specify the modem initialization string to be used when the driver starts. A typical string is ATZ, which clears the modem registers for most modems.
Error Log	Specify the error file name. This is optional.
Baud	Click to open a menu of bandwidth rate options.
Bits	After the start bit has been sent, the transmitter sends the actual data bits. Menu choices are 5, 6, 7, or 8 data bits. Both the receiver and transmitter must agree on the number of data bits, and the baud rate. Most devices transmit data using 7 or 8 data bits.
Parity	A method of detecting errors in transmission. Click to open a menu of parity bit choices: None, Odd, or Even.
Require carrier	Select this to require the carrier signal online before connecting.
Use RTS/CTS	Select this to use hardware flow control.
Use XON/XOFF	Select this to use software flow control.
Use 8 Bit	Select this to enable <code>ISTRIP</code> .

Auto-Reconnect Options pane

This table shows the parameters in the Auto-Reconnect Options pane:

Parameter	Description
Auto-reconnect	Select this to automatically reestablish a broken connection.
Reopen Time	Specify the minimum number of seconds to wait after connection failure before reconnection. The default is 5. This option is available only when Auto-reconnect is selected.

Start-up Procedures pane

Click **Edit** to open the **TPS Editor**. Use this to select the procedures to run when the thread starts. This is a TPS (Tcl Procedure Stream). Use it to trade messages with the remote end.

Configuration keys

The engine constructs use these keys to configure PDL Async with configuration lists:

- TYPE: async
- NAME: PDL name
- BAUD : baud rate
- PARITY: parity
- DATA-BITS: word size
- PATH: /dev/ttyname
- REQUIRE-CD: Require Carrier Detect (boolean)
- HW-FLOW-CONTROL: Use RTS/CTS flow control (boolean)
- XON-XOFF: Use software flow control (boolean)

PDL TCP/IP protocol

This is a Transmission Control Protocol/Internet Protocol implemented with PDL (Programmable Driver Language).

As with the TCP/IP driver, a PDL can act as server or client. Clients must specify the remote host and port/service with which to connect. servers specify only the port/service on which to await connections.

As the PDL/Tcl code identifies message boundaries, do not use length-encoding.

PDL Options and TCP/IP Options panes

This table shows the parameters on the PDL Options and TCP/IP Options panes:

Parameter	Description
PDL	Click List to select the PDL script name.
Client	Select this for the site to make a connection as a client of another TCP/IP process.
server	Select this for the hub to provide a connection point for TCP/IP clients.
Multi-server	Select this to have multiple clients simultaneously connect to one port. Click Configure to open the Multi-server Options dialog box. Use this dialog box to configure the multi-server attributes.
Host	Click List to open the Hosts List dialog box, where you can select the host name for this connection. This option is unavailable when server is selected.

Parameter	Description
Port	<p>Click Select to open the Select Port dialog box. Use this dialog box to select the server port.</p> <p>When Type is Client, Port is List.</p> <p>When Type is server or Multi-server, Port is Select.</p>
Local Binding Address	<p>Specify the IP address or a resolvable host name, or click List to make a selection.</p>
TCP Connection Timeout	<p>This is the time-out for the PDL/TCP-IP client connecting to the server.</p> <p>In the NetConfig file, this is listed as TCP_CONNECTION_TIMEOUT.</p> <p>To connect to the server, the PDL/TCP-IP thread invokes <code>connect()</code>.</p> <p>Use this option to configure the wait time. This can also be configured using the <code>CL_TCPIP_POLLING</code> environment variable. The polling interval is in the 50-2000 ms range. The default value is 200 ms.</p> <p>This affects:</p> <ul style="list-style-type: none"> • TCP-IP • PDL/TCP-IP • Intersite ICL
IPv4/v6 Dual	<p>If this mode is enabled and the server is bound to "0.0.0.0" and ":::", then one extra listening socket is created for IPv6. Local Binding Address is blank. Two sockets are created: one for IPv4 and one for IPv6, both of which are handled separately.</p>

Parameter	Description
SSL	<p>Select this for SSL (Secure Socket Layer) configuration.</p> <p>Click Configure to open the SSL dialog box.</p> <ul style="list-style-type: none"> • Mode choices reflect which Type option was selected: Client or server. All clients connecting to a multi-server must have the same configuration, that is, all must be SSL or non-SSL. • SSL Protocol is used to select the openssl version from the menu. When you select an SSL protocol, a description of the selected protocol is shown in the comment field together with the Mode description. • SSL Cipher Suites Set ciphers in this field. If no cipher is set, then the default cipher suites are used. If this field is not set, then the default cipher suites are used. Anonymous mode: server: !DEFAULT:HIG:ADH Client: ALL Non-anonymous mode: HIGH:RC4+RSA:+MD5:!DHE:!3DES:!EXP:!ADH:!AES256-SHA:!AES128-SHA:!EDH:!aNULL:!eNULL:!NULL

Data Options pane

This table shows the parameters on the Data Options pane:

Close after write	Select this to close the connection after a write.
Use DRIVERCTL control	<p>Select this to have the driver examine DRIVERCTL for control keys such as CLOSE.</p> <p>The TCP/IP PDL driver handles one driver control directive. It is passed by an outbound message, and is defined in DRIVERCTL in the message metadata. The value to the directive is boolean.</p> <p>This directive is CLOSE. This closes the connection. For example:</p> <pre>msgmetaset \$mh DRIVERCTL "{CLOSE 1}"</pre>

Auto-Reconnect Options Pane

This table shows the parameters on the Auto-Reconnect Options pane:

Auto-reconnect	Select this to automatically reestablish a broken connection.
Reopen Time	Specify the minimum number of seconds to wait after connection failure before reconnection. The default is 5. This option is available only when Auto-reconnect is selected.
Delay connection until needed	To use this feature, select Delay connection until needed and Close after write , and clear Auto-reconnect . The delayed connection feature overrides auto-reconnect all of the time. See Delay connection until needed .

Start-up Procedures pane

Click **Edit** to open the **TPS Editor**. Use this to select the procedures to run when the thread starts. This is a TPS (Tcl Procedure Stream). Use it to trade messages with the remote end.

Delay connection until needed

To use this feature in PDL TCP/IP, select **Delay connection until needed** and **Close after write** on the **PDL TCP/IP Protocol Properties** dialog box and clear **Auto-reconnect**. The delayed connection feature overrides auto-reconnect all of the time.

When selected, connection allocation is delayed until there is a message to write. By default, the check box is not selected. When using this option, the driver allocates a connection only when required for an outbound message.

To use this feature in TCP/IP, select **Delay connection until needed** and **Close after write** on the **TCP/IP Protocol Properties** dialog box and clear **Auto-reconnect**. The delayed connection feature overrides auto-reconnect, with one exception:

If **Delay connection until needed** and **Auto-reconnect** are both selected, then reconnect is still in effect, even if there is a transaction error.

When selected, connection allocation is delayed until there is a message to write. By default, the check box is not selected. When using this option, the driver allocates a connection only when required for an outbound message.

TCP/IP protocol

TCP/IP (Transmission Control Protocol/Internet Protocol) is a transmission protocol and addressing mechanism that creates a reliable, ordered, stream-oriented data transmission between two network peers.

Use the TCP/IP protocol driver to interface with systems communicating by TCP/IP and using length-encoding as the method for bounding messages. The driver supports several methods for separating/delimiting messages sent across the connection.

This table shows the methods for separating/delimiting messages:

Method	Description
Driver message separation	TCP/IP connections operate in the stream mode. The protocol trades characters between end-points. The end-point applications, the system and the remote system, must use a common delimiter scheme. The driver makes no attempt to separate one logical message from another.
Inbound messages	The remote application can deliver messages to the TCP/IP driver at any time. When the driver receives a message, it passes it into the engine for processing.
Outbound messages	When a message is delivered to the driver for writing, the driver transmits any length-encoding information followed by the message body.

TCP/IP Options pane

This table shows the parameters in the TCP/IP Options pane:

Parameter	Description
Client	Select this for the Hub to make a connection as a client of another TCP/IP process.
Server	Select this for the Hub to provide a connection point for TCP/IP clients.
Multi-Server	Select this to have multiple clients simultaneously connect to one port. Click Configure to open the Multi-Server Options dialog box. Use this dialog box to configure the multi-server attributes.
Host	Click List to open the Hosts List box. Use this list box to select the host name for this connection. This option is unavailable when Server is selected.

Parameter	Description
Port	<p>Click Select to open the Select Port dialog box. Use this dialog box to select the Server port.</p> <p>See Server Port configuration.</p> <p>When Client is selected, Port is List.</p> <p>When Server or Multi-Server is selected, Port is Select.</p> <p>The TCP/IP Host and Port configuration elements provide the primary description for the connection, where:</p> <ul style="list-style-type: none"> • Host identifies the TCP/IP entity on the network (CPU). • Port identifies a particular networking endpoint on the machine. • If the driver is configured as a Client, then it opens a connection on the selected port on the host machine. • If it is configured as a Server, then it awaits a connection on the port on the local machine that is listed in Host.
Local Binding Address	<p>Specify the IP address or a resolvable host name, or click List to make a selection.</p>
TCP Connection Timeout	<p>To connect to the server, the TCP-IP thread invokes <code>connect()</code>.</p> <p>Use this option to configure the wait time. This can also be configured using the <code>CL_TCPIP_POLLING</code> environment variable. The polling interval is in the 50-2000 range. The default value is 200.</p> <p>This affects:</p> <ul style="list-style-type: none"> • TCP-IP • PDL/TCP-IP • Intersite ICL
IPv4/v6 Dual	<p>When IPv4/v6 dual mode is enabled and the server is bound to "0.0.0.0" and ":::", one extra listening socket is created for IPv6. Local Binding Address is blank. Two sockets are created: one for IPv4 and one for IPv6, both of which are handled separately.</p>

Parameter	Description
SSL	<p>Select this for SSL (Secure Socket Layer) configuration. When running under security, SSL (Secure Socket Layer) connections let messages pass through or from servers with encryption capabilities. Click Configure to open the SSL dialog box.</p> <p>The Mode choices reflect which option was selected: Client or Server.</p> <p>All clients connecting to a multi-server must have the same configuration, that is, all must be SSL or non-SSL.</p> <ul style="list-style-type: none"> Select the mode from the Mode menu. SSL Protocol is used to select the openssl version from the menu. When you select an SSL protocol, a description of the selected protocol is shown in the comment field together with the mode description. SSL Cipher Suites Set ciphers in this field. If no cipher is set, then the default cipher suites are used. If this field is not set, then the default cipher suites are used. Anonymous mode: Server: !DEFAULT:HIG:ADH Client: ALL Non-anonymous mode: HIGH:RC4+RSA:+MD5:!DHE:!3DES:!EXP:!ADH:!AES256-SHA:!AES128-SHA:!EDH:!aNULL:!eNULL:!NULL

Data Options pane

This table shows the parameters in the Data Options pane:

Parameter	Description
ASCII	<p>For Length, specify an integer for the message size in bytes. For example, a length of 2 means a message size less than 99 bytes; a length of 8 means a message size less than 99999999 bytes.</p> <p>For Fill, specify the fill character.</p> <p>For Justification, select Left or Right.</p> <p>For Length Calculations, select Inclusive to include length-encoding bytes, or Exclusive to exclude length-encoding bytes.</p>

Parameter	Description
Binary	<p>For Length, click the arrow to select the number of bytes from the menu.</p> <p>For Order, select Native or Network.</p> <p>For Length Calculations, select Inclusive to include length-encoding bytes, or Exclusive to exclude length-encoding bytes.</p>
Encapsulated	<p>Use this for configuring the HL7 MLP (Minimum Layer Protocol). After selecting this option, click Configure to open the Encapsulated Options dialog box.</p> <p>See TCP/IP MLP.</p>
Close after write	Select this to close the connection after a write.
Use DRIVERCTL control	Select this to have the driver examine DRIVERCTL for control keys such as CLOSE, and populate DRIVERCTL when events such as a socket close happen.
Write 0-length messages	Select this to write nothing to the connection.

Auto-Reconnect Options pane

This table shows the parameters for the Auto-Reconnect Options pane:

Parameter	Description
Auto-reconnect	Select this to automatically reestablish a broken connection.
Reopen Time	Specify the minimum number of seconds to wait after connection failure before reconnection. The default is 5.
Delay connection until needed	<p>To use this feature, select Delay connection until needed and Close after write, and clear Auto-reconnect.</p> <p>See Delay connection until needed.</p>

Start-up Procedures Pane

Click **Edit** to open the **TPS Editor**. Use this to select the procedures to run when the thread starts. This is a TPS (Tcl Procedure Stream). Use it to trade messages with the remote end.

Multi-Server options

Select **Multi-Server** to have multiple clients simultaneously connect to one port.

Then, click **Configure** to open the **Multi-Server Options** dialog box. Use this dialog box to configure the multi-server attributes. This helps to balance engine traffic by limiting incoming connections.

This table shows the parameters for the **Multi-Server Options** dialog box:

Parameter	Description
Maximum Clients	Specify the maximum number of clients that are accepted at one time. Additional clients (connections) are refused if this value is exceeded.
Maximum Outbound Queue Depth	Specify the maximum queue depth that is accepted. Additional clients (connections) are refused if this value is exceeded.
Maximum Pre-Translate Queue Depth	Specify the maximum queue depth that is accepted. Additional clients (connections) are refused if this value is exceeded. Values that are less than or equal to zero are interpreted as unlimited.
Save client IP and port to driver control	Select this to copy the client's IP and port into the driver control string along with the connection ID.

Active client list

The engine maintains a file in the process directory with one line containing a list of active clients. This file contains a space-separated list of connection IDs. If **Save client IP and port to driver control** is selected, then this file contains space-separated lists of {connection ID, IP address, port}.

For example:

```
{10 10.30.10.35 9876} {15 10.30.10.37 4322} {20 10.30.10.37 4323}
```

This file, named `thread_name.cli`, is suitable for reading as a Tcl list. The information in this list is maintained also in the `client_list` Tcl global variable.

```
{CONNID 0} {IPVERSION 4} {CLIENTIP 127.0.0.1} {CLIENTPORT 4191}
```

Connection ID

Clients are identified by a unique connection ID that exists for the duration of the connection and is not reused. This connection ID is encoded into the DriverControl metadata with the key `CONNID`. This field is set based on the client for all inbound messages.

Reply-type messages automatically get the `CONNID` of the original outbound message. You can also set the `CONNID` using UPoCs. If the `CONNID` for an outbound message is not valid, then the message fails sending and transitions to the error database or send fail procedure.

TCP/IP DRIVERCTL options

The TCP/IP driver handles three driver control directives. They are passed to it by an outbound message, and are defined in `DRIVERCTL` in the message metadata. All are boolean.

These directives are:

- `CLOSE`: Closes the connection.
- `RECONNECT`: Reconnects the connection.
- `WRITEZERO`: Writes nothing to the connection.

For example:

```
msgmetaset $mh DRIVERCTL "{TCP/IP {{CLOSE 1} {WRITEZERO 1} }}"
```

The driver also creates an empty message with `close` defined if its peer closed the connection.

Driver control when not using multi-server

Driver control remote connection information is also created when not using multi-server. There are no controls for users to turn this off/on or control the content.

In this case, driver control always has the `CONNID` as zero and is a keyed list that looks similar to:

```
{CONNID 0} {IPVERSION 4} {CLIENTIP 127.0.0.1} {CLIENTPORT 4191}
```

Length encoding

This assumes that all TCP/IP processed messages are length-encoded.

Prefix all messages received from a TCP/IP connection with a 2- or 4-byte integer value that specifies the message length. This value is stripped from the message before translation.

Prefix all messages sent to a TCP/IP connection with a 2- or 4-byte integer value.

Socket addresses

Sockets provide an endpoint for communications. With TCP/IP, sockets are not tied to a destination address. Applications sending messages can tie the socket to a specific destination address for the duration of the connection. They can also specify another destination address for each message.

TCP/IP creates socket addresses that are unique throughout all Internet networks. To derive the Internet socket address, TCP/IP concatenates the Internet address of the local host interface with the port number.

Because the internet address is always unique for a particular network host, an individual socket's address on a particular host is also unique.

Because each connection is defined by the pair of sockets it joins, every connection between internet hosts is also uniquely defined.

Port numbers

TCP/IP provides a set of 16-bit port numbers within each host. Because each host assigns port numbers independently, ports on different hosts can have the same number.

When a client process requires a particular service at a host, the client process sends a service request. This is sent to the socket address for the host port.

If a process on the host is listening, then the server process does one of:

- Services the request using the port.
- Transfers the connection to another port that is temporarily assigned for the duration of the connection to the client.

Using temporarily assigned, or secondary, ports frees the original port for the host port to concurrently handle additional requests.

This table shows the port number services:

Port range	Services
0 to 023	Reserved for root processes.
0 to 255	Reserved for official Internet services.
256 to 1023	Reserved for common Internet network services. The <code>/etc/services</code> file lists the common port numbers.
1024 +	Used by processes requiring a temporary port after an initial service request is received.
5000 +	Suggested port numbers for users.
10300	The TCP/IP port number of the GDBM server listening port.

TCP/IP MLP

HL7 Minimum Layer Protocol (MLP), also called Minimum Lower Layer Protocol (MLLP), is a basic encapsulation method for the transmission of HL7 messages.

The encapsulation is defined as a start block, the message, and an end block.

- The start block consists of a single byte with a value of 0x0B. This is the ASCII VT character code.
- The end block consists of two characters with the values 0x1C followed by 0x0D. This is the ASCII characters codes for FS and CR.

Encapsulated data type on TCP/IP Protocol Properties dialog box

To use MLP, select **Encapsulated**, then select **Configure**. This opens the **Encapsulated Options** dialog box.

This table shows the parameters on the **Encapsulated Options** dialog box:

Parameter	Description
MLLP	<p>This is the default. The Start Block, End Block, Commit ACK, and Negative ACK options are disabled, so that the standard MLLP Start Block and End Block are used.</p> <p>The default Timeout is 30 seconds and the corresponding Timeout Handling is RESET. Both are selected automatically if no user-specified settings are loaded.</p>
MLLP2	<p>When this is selected, the Start Block, End Block, Commit ACK, and Negative ACK options are disabled, which means the related MLLP2 standard values are used.</p>
USER	<p>When this is selected, the Start Block is enabled and is empty by default. End Block is also enabled and is populated with "\n," which means a new line separator is set by default.</p> <p>Start Block can be empty. End Block must have at least one byte. That is, one ASCII character.</p> <p>A user-specified Start Block and End Block can have up to 31 bytes. Lengths over this causes an error.</p>
USER2	<p>When this is selected, all components are enabled and editable.</p> <p>Start Block can be empty. End Block must have at least one byte. That is, one ASCII character.</p> <p>A user-specified Start Block, End Block, Commit ACK, and Negative ACK can have up to 31 bytes. Lengths over this causes an error.</p>

Timeout and Timeout Handling

The default time-out is 30 seconds and the corresponding default time-out handling is **RESET**. Only integers equal to or greater than -1 are accepted.

A time-out of -1 implies "none." **Timeout Handling** is disabled in this case.

Timeout Handling can also be **close**. This specifies that the remote connection is to be closed.

- For the server mode, it is up to the remote connection to reconnect to continue normal operations of message transfer.
- For the client mode, the connection is closed and an error status is returned to the engine.

The **Timeout** and **Timeout Handling** options apply to all modes. If there are no time-out settings in NetConfig, then it works the same as the default settings.

UPoC protocol

The UPoC driver uses Tcl Procedure Streams to run Tcl commands and pass messages into the engine. As messages are delivered to the driver, they go through an outbound TPS for delivery to the database.

This driver runs a TCL script on a user-specified interval, usually to read messages into the interface. Alternately, the user can configure it to run a TCL script when a message write is attempted.

You can use an API to write code fragments in Java for UPoCs. Although there are many UPoCs, there are only three interface styles between the engine and the user code:

- **TPS:**
Controls/extends message movement through the engine.
- **TrxID:**
Determines a transaction code, or routing identifier, for a given message.
- **XLT:**
Transforms certain fields within a message as part of a transformation specification.

In the Java API, each of these styles is represented as a class which can be extended by the interface developer.

Options pane

This table show the parameters for the Options pane:

Parameter	Description
Read TPS	<p>Click Edit to open the TPS Editor. Then, click Add to select the inbound procedures. (IB disabled if 0 procs).</p> <p>The driver's inbound (read) TPS generates inbound messages and passes them into the engine (READTPS).</p> <p>This option is disabled if Use advanced scheduling is selected.</p> <p>Messages returned with CONTINUE disposition are treated as inbound messages.</p> <p>When generating a message in Start mode in Read TPS, inbound messages are placed in the IB Pre/Post queue twice. This leads to an error in translation. To prevent this, do not generate messages in Start mode in Read TPS.</p>
Read Interval	<p>Specify the inbound interval. This is the READINTERVAL. The IB is disabled if there are no procs. This option is disabled if Use advanced scheduling is selected.</p>
Use advanced scheduling	<p>Select this to schedule recurring events, such as recycling log or SMAT files. Then, click Setup to open the Scheduling dialog box.</p>

Parameter	Description
Write TPS	<p>Click Edit to open the TPS Editor. Then, click Add to select the outbound procedures. The driver's outbound (write) TPS is required to write the message, using <code>WRITETPS</code>.</p> <p>This is run when an outbound message is available for protocol delivery.</p> <p>In the UPoC Write TPS, only use <code>CONTINUE</code> (success) or <code>ERROR</code> (failure).</p> <p>A <code>CONTINUE</code> disposition goes to <code>SEND OK</code>.</p> <p>When you use the <code>CONTINUE</code> disposition your messages out should increment and if you have a procedure applied, the Send OK TPS is run.</p> <p>When you use <code>ERROR</code>, the message is sent to the Send Fail TPS. If there is no code in Send Fail TPS or it returns a <code>CONTINUE</code>, then the message is re-queued at the top of the queue for re-delivery. If a retry is not necessary, then use <code>ERROR</code> inside the Send Fail procedure.</p> <p>Use the Send OK/Fail procs to control the post delivery aspects of your UPoC protocol.</p> <p>Note: For your Send Fail procedure to run, the Retries setting on the Outbound tab must be greater than 0.3 or 10 is typical, depending on the interfaces requirements.</p>

Start-Up Procedures pane

Click **Edit** to open the TPS Editor. Use this to select the procedures and Java class to run when the thread starts. This is a TPS (Tcl Procedure Stream). Use it to trade messages with the remote end.

Inbound messages

If a non-zero inbound scan-time value is provided, then the driver runs its inbound (read) TPS according to other threads' actions. It is run no less than N seconds between invocations. Unlike most TPS instances, there is no input for this.

Because there is no input, the UPoC driver introduces another time-based mode. Similar to start-up, procedures may or may not receive a `MSGID` in their arguments. This mode is distinctive from start-up mode so that procedures can take other actions.

No matter how the TPS is called, it produces zero or more messages under these dispositions:

- `CONTINUE`:
Passes into engine and places on inbound pre-TPS queue
- `ERROR`:

Transitions to error database

- **KILL:**
Destroys the message
- **OVER:**
Illegal; warns and destroys the message
- **PROTO:**
Places on outbound post-TPS queue
- **SEND:**
Places on inbound post-TPS queue

Outbound messages

The outbound operation of this driver pushes messages through the TPS.

The dispositions that are procured by the TPS are:

- **CONTINUE:**
Destroys
- **ERROR:**
Transitions to error database
- **KILL:**
Destroys the message
- **OVER:**
Illegal; warns and destroys the message
- **PROTO:**
Places on outbound post-TPS queue
- **SEND:**
Places on inbound post-TPS queue

Prosper Async protocol (UNIX)

Prosper Async is an asynchronous serial interface. For asynchronous communications, specify the behavior characteristics such as baud rate, number of data bits, parity, and other characteristics.

The Prosper Async protocol driver communicates by the Prosper protocol over multiple TTY devices. This contains information for managing both the input and output sides of the driver.

Options pane

This table shows the parameters in the Options pane:

Parameter	Description
Line	The first device name. For example, tty0.
# Lines	The number of lines on which to listen.
Baud	Click to open a menu of bandwidth rate options.
Parity	A method of detecting errors in transmission. Click to open a menu of parity bit choices.
Data Bits	The number of data bits in each character. Usually, this is 7 or 8.
Stop Bits	The number of stop bits at the end of every character.
Terminal Msg Size	The maximum terminal packet size.
Host Msg Size	The maximum host packet size.
NAK count	The maximum number of NAKs to send/accept.
NAK wait	The number of seconds between successive NAKs.
ENQ wait	The number of seconds between successive enquiries.
Audit file	The audit log file name.

Start-up Procedures pane

Click **Edit** to open the **TPS Editor**. Use this to select the procedures to run when the thread starts. This is a TPS (Tcl Procedure Stream). Use it to trade messages with the remote end.

Link Async protocol (UNIX)

Link Async is an asynchronous serial interface. For asynchronous communications, specify the behavior characteristics such as baud rate, number of data bits, parity, and other characteristics.

The Link Async protocol driver communicates by the Link protocol over multiple TTY devices. It contains information for managing both the input and output sides of the driver.

Options pane

This table shows the parameters in the Options pane:

Parameter	Description
Line	The first device name. For example, tty0.
# Lines	The number of lines on which to listen.

Parameter	Description
Baud	Click to open a menu of bandwidth rate options.
Parity	A method of detecting errors in transmission. Click to open a menu of parity bit choices.
Data Bits	The number of data bits in each character. Usually, this is 7 or 8.
Stop Bits	The number of stop bits at the end of every character.
Terminal Msg Size	The maximum terminal packet size.
Host Msg Size	The maximum host packet size.
NAK count	The maximum number of NAKs to send/accept.
NAK wait	The number of seconds between successive NAKs.
Audit file	The audit log file name.

Start-up Procedures pane

Click **Edit** to open the **TPS Editor**. Use this to select the procedures to run when the thread starts. This is a TPS (Tcl Procedure Stream). Use it to trade messages with the remote end.

Advanced scheduling

Advanced scheduling uses the Network Configurator to enter Fileset Local, Fileset FTP, Database, HTTP Client, Java, and UPoC schedules. This is in terms of absolute dates and time, instead of offset seconds. Schedules can be set for backups, alerts, file transfers, and so on.

When selected, advanced scheduling is used in place of interval-based scheduling for the thread being configured.

Caution: Exercise caution when using advanced scheduling, as you could create schedules that request impossible behavior from the engine.

Absolute time entry circumvents the problem of using an offset time if the thread is restarted. For example, if an offset seconds equivalent of 12 hours is entered and the thread is restarted, the event happens 12 hours later.

Each event that is listed under a schedule has an entry similar to crontab. This specifies the recurrence times, a description, and, in the case of UPoC, a Tcl procedure stack and corresponding arguments.

After the Network Configurator obtains the protocol selection and timer event schedules from the user, it generates a Tcl list containing this schedule. This list is stored in the NetConfig file. The engine extracts the scheduling information from this Tcl list, and registers and updates the timer events.

Because scheduled events are recurring events, certain commands are provided to check the status and to disable and enable individual events or entire schedules.

Note: User discretion is required when using advanced scheduling. If the running of a Fileset or UPoC activity takes a long time, then the scheduling of the next event could be missed.

For example, several files take ten minutes to be read by the engine using the Fileset protocol. These events are scheduled every five minutes. At least one activity does not get scheduled. This is because the next event is scheduled after the last event has finished running. Only one event is scheduled at a time per thread.

If advanced scheduling is set, then the timer event is run only when advanced scheduling is triggered. Fileset-local and Fileset-FTP protocol threads do not immediately run the event when the thread starts. This also applies to the database and Java driver protocols.

Scheduling dialog box

After selecting **Use Advanced Scheduling**, click **Setup** to begin.

- Click **New** to configure new events for a schedule.
- Click **Edit** to edit an existing event definition.
- **Description** shows an event name that is propagated to a log file if the event causes an error.
- **Recurrence** shows an entry that describes when an event should happen.
- **Procs** lists the names of any procs that are to be run by the event.

Network Configurator schedule configuration

Use the Network Configurator protocol dialog boxes to specify the event times and dates.

If event scheduling is required for the Database, Fileset Local, Fileset FTP, HTTP Client, Java, or UPoC protocols, then select **Use Advanced Scheduling**. Then, click **Setup** when setting up the individual protocol in Network Configurator.

For the Fileset and Database protocols, advanced scheduling uses **Scan Interval**.

For the UPoC protocol, advanced scheduling uses **Read Interval**.

For the HTTP Client and Java protocols, advanced scheduling uses **Query Interval**.

Advanced scheduling replaces interval-based scheduling. A schedule defined from setup disables the interval. For Fileset, this is the **Scan Interval**. For UPoC, this is the **Read Interval**. For HTTP Client, this is the **Query Interval**.

If the engine finds a schedule to use, then it ignores the interval setting.

Note: When configuring advanced scheduling, use procs for the UPoC and HTTP Client protocols. Procs are not available for use in Fileset and Database protocols.

Event properties

Click **New** or **Edit** on the **Scheduling** dialog box to open the **Event Properties** dialog box. Use this dialog box to define or redefine an event for a schedule.

Each field contains a user-defined value that defines the event.

Entries in these fields can be any of these:

- Single number
- Pair of numbers separated by a dash to indicate a number range
- Comma-separated list of numbers and ranges
- Asterisk (a wildcard representing all valid values for that field). Remove the asterisk if the entry is a number.

Fields include:

Field	Description
Description	A description of the event.
Seconds	A value in the range 0-59.
Minutes	A value in the range 0-59.
Hours	A value in the range 0-23, where 0 is midnight.
Days of Month	A value in the range 1-31.
Months	A value in the range 1-12.
Days of Week	A value in the range 0-6, where 0 is Sunday.
TPS	Click Procs to open the TPS Editor, where procedures can be attached to an event. The TPS option is available only when using the HTTP Client or UPoC protocol.

Event recurrence

User-defined recurrence values are shown on the **Scheduling** dialog box, replacing the asterisks in the same order that is shown on the **Event Properties** dialog box.

Each asterisk holds a particular place value:

*	*	*	*	*	*
<i>seconds</i>	<i>minutes</i>	<i>hours</i>	<i>day of month</i>	<i>month</i>	<i>day of week</i>

Step values

Specify step values using the **/n** suffix with any number. For example, the string **8-18/2** in **Hours** means "every two hours from 8 AM to 6 PM."

The specification of days can be made in two fields: **Days of Month** and **Days of Week**. If both are specified in an entry, then they are cumulative. For example, **0 0 0 1,15 * 1** specifies that a task will run at midnight on the first and fifteenth of every month, and every Monday.

The year is not specified. The current year is assumed unless the time of the next event is in the past. When this happens, the second, minute, hour, day-of-month, or year is incremented to the next recurrence that is in the future.

Note: These entries are recurring events. There is no provision to specify a single event happening only once.

Configuring a new schedule sequence

- 1 In the Network Configurator, on the **Properties** tab, click **Properties** by the **Protocol** field.
Note: Only the Fileset Local, Fileset FTP, HTTP Client, and UPoC protocols use Advanced Scheduling.
- 2 To access the **Scheduling** dialog box, select **Advanced Scheduling** and click **Setup**.
Note: For an HTTP Client, you must select **Time Driven** from the **Driver Mode** list to enable advanced scheduling.
- 3 Click **New**.
This example creates a schedule for running a Tcl procedure named "reinitialize" with these parameters:
 - 6AM to 5PM
 - Monday through Friday
 - Every two hours
 - To be run at nine minutes, zero seconds after the hour
- 4 In the **Description** field, specify a description for the event.
- 5 Fill in the fields to specify event times and dates.
See [Event properties](#).

Caution: Exercise caution when configuring parameter values. For example, if the **Seconds** field is left without any value, then the procedure attempts to run every second. To avoid this, specify appropriate values for all parameters, especially **Seconds** and **Minutes**.

- 6 For the **TPS** field, click **Procs** to open the TPS Editor.
The **TPS** field is not editable. You must select a procedure from the TPS Editor. The **TPS** option is available only when using the HTTP client or UPoC protocol.
- 7 Click **Add** to select a procedure.
If you must create a custom procedure, then first use the Script Editor.
To make a selection, select the procedure from the list. If you have created a custom procedure, then it is displayed in the list. In this example, **reinitialize** is selected.
- 8 Click **OK** on the **TPS Properties** dialog box.

This adds the procedure to the TPS Editor.

- 9 Click **OK** to close the TPS Editor.

This adds the procedure to the TPS field on the **Event Properties** dialog box.

- 10 Click **OK**.

The new event is now listed on the **Scheduling** dialog box.

- 11 Click **OK**.

Modifying an existing schedule

- 1 Open the **Protocol Properties** dialog box for the thread to modify and click **Setup** next to **Advanced Scheduling**.
- 2 Highlight the event to modify and click **Edit** (or double-click the event).
- 3 Modify the event properties.
In this example, instead of having the procedure run at nine (9) minutes past the hour, it is run at 19 minutes past the hour. On the **Event Properties** dialog box, specify **19** for **Minutes**.
- 4 Click **OK** on the **Event Properties** dialog box to confirm the modification.
- 5 Click **OK** on the **Scheduling** dialog box to accept the new schedule and close the dialog box.
- 6 Save the file.

Network Monitor timer thread state

The Network Monitor is used to monitor and control the state of processes and threads. It can also be used to monitor a timer thread state.

A clock icon is used to indicate that the thread is running according to a schedule.

Note: Event configuration or modification is accomplished through the Network Configurator.

- 1 Right-click the scheduled thread icon on the Network Monitor.
- 2 Select **View Schedule**.

This opens the **Scheduling** dialog box, from which you can enable or disable one or more listed events. This option is available only when timed events are configured.

Script Editor

The **Script Editor** is a built-in editor that you can use to create, open, and save scripts through the host server.

You can use the **Script Editor** to customize many aspects of the engine and directly affect how messages are handled. It does this by creating files for storing Tcl procedures, and editing existing files or Tcl procedures.

Tcl scripts can be expanded in the Site Manager to list the procedures that they contain. Double-clicking a file opens it in the **Script Editor**.

Points to remember:

- When doing a search, **Shift+Enter** performs a backwards search.
- Searches can include more than one word.
- Searches are case insensitive.

Additional features are also available. See [Advanced features](#).

Script file locations

Custom Tcl procedure files reside in `$HCISITEDIR/tclprocs`.

JavaScript scripts reside in `$HCISITEDIR/scripts`.

Python scripts reside in `$HCISITEDIR/scripts`.

Script Editor and Tcl procedures

Custom scripts affect engine behavior using Tcl procedures. Configure these procedures to run at certain UPoCs (User Points of Control).

Tcl procedures within the engine

A Tcl Procedure Stream (TPS) lists Tcl procedures that are used in sequence. It uses the TPS binary to invoke the procedures in series with the correct arguments. This is the most prevalent form of UPoC Tcl code.

Trx ID determination decides a message's transaction, as opposed to offset/length or variant. After determining the transaction ID, the translation thread translates and routes the message appropriately.

A XLT (translation) code fragment calls procedures that receive their input through a certain set of variables, and returns data through another set. Any values that parameters return are ignored.

Tcl procedures outside the engine

Two types of UPoC Tcl code are used outside of the engine for testing purposes:

- TPS-test end procedures
Uses end procedures to process messages produced by the Testing Tool. Used with `hcitpctest`.
- Route-test end procedures
Uses end procedures to process messages produced by the Testing Tool. Used with `hciroutetest`.

Selecting a Tcl procedure

To select Tcl code:

- 1 Click a category and its available functions display in the function list.
- 2 Select a function and its description displays in the text box at the bottom of the dialog box.
- 3 Double-click the function to place it in the **Script Editor**.

Advanced features

All resources are stored in %HCIR00T%\client\ScriptEditor.

Additional features include:

- Multiple Script Editors can be simultaneously open for several files.
- Tcl keywords, braces, and strings are shown in several colors.
For example, Tcl keywords are displayed in blue color, and braces/brackets are in red color. Clicking a brace/bracket also selects the corresponding brace/bracket.
- On the right-click menu, selecting **Add Proc** opens the **Add Proc** dialog box, where you specify a **Proc Name** and select a template. This includes all proc types. The new proc is appended to the end of the current Tcl file.
See [Proc types](#).
- Clicking **Folding** on the right-click menu opens a menu of expand/collapse code folds.
- The Tcl language is syntax-related, and assists you in checking for errors, misspellings, mismatched braces, and so on. This can be helpful for users (Tcl programmers) doing coding.
- The TclKeywords folder stores all auto-complete help resources, and all are named with a Tcl keyword.
 - The _fileIndex folder stores all keywords line by line.
 - The themes folder stores all theme definition files. These control the code foreground, background, and highlighting colors displayed in Script Editor. By default, Script Editor loads default.xml.
To use another theme, rename the original default.xml file and then rename the other template file as default.xml.
 - TclCodeTemplates.xml stores all code templates. In this file, the \${cursor} for each template is a placeholder for the cursor. This is where the cursor is located when a code template is inserted.
Templates are disabled by setting enabled="false".
- Syntax highlighting and brace/bracket/single quote/double quote matching are available.
For example, you can use the matching bracket feature to find the end of a while loop.
- Line numbers provide a method for locating syntax errors. These numbers are located on the left-hand and right-hand sides of the Script Editor.
- A highlight bar is along the right side of the editor that indicates where matches are located in the file.
- When you type characters in the Script Editor, the **Auto-Complete** window opens to show a matched Tcl keywords list.

Client Preferences Script Editor options

The **Client Preferences** dialog box has a **Script Editor** tab where you can configure the **Script Editor**.

This table shows the available configuration options:

Option	Description
Font Size	This is an integer from 7 to 32, which controls the font size of the displayed code within the editor.
Tab Size	This is an integer from 1 to 32, which controls the width of the Tab key displayed within the editor.
Emulate tabs with spaces	When this is selected, and you press Tab to insert a tab in the code, the editor auto-converts the Tab key into spaces.
Activate Auto-Complete	When this is selected, the Auto-Complete window is enabled. If this is cleared (default), then the Auto-Complete window does not open unless you press Ctrl+Space .
Open Auto-Complete Window	This is the shortcut key to manually open the Auto-Complete window. The default is Ctrl+Space . This windows opens only when there are choices to select from. For example, entering a "j" auto-completes with "join" without the Auto-Complete window opening. Entering an "f" opens the Auto-Complete window with a list of choices. The auto-complete functionality supports standard TCL, JavaScript, and Python.
Fill with Code Template	This triggers the code template function. For example, if you specify "for" and then press Ctrl+Shift+Space , the editor automatically inputs the code template for the "for" loop.
Jump to Matching Bracket	This is useful for bracket matching, especially when the starting and ending bracket extend across many lines. To use this function, position the cursor immediately after a bracket, then press the shortcut keys. This moves the cursor to the corresponding matching bracket. The Script Editor scroll bar moves to display the line to which the cursor jumped. The default is Ctrl+J .

Proc types

Click **Add Proc** to open the **New Proc** dialog box. The **New Proc** dialog box is also available from the right-click menu. Click the **Proc Type** arrow to select a procedure-type template from the menu.

This table shows the available templates:

Template	Description
code_set	<p>A HIPAA code set Tcl procedure template that gives an example of querying a lookup table for a particular key. This uses the <code>dfлт=</code> specification in the lookup table. This determines the value to be returned if there is no table entry matching the input key value.</p> <p>This default value is set to <code>_non-existent_</code> in all HIPAA code set lookup tables.</p>
ibdir	A TPS template that gets the IBDIR path from NetConfig. This accepts a TPS keyed list that contains the keys MODE, MSGID, and ARGS, and returns a TPS disposition list.
ibin	A TPS template that parses ibmime and retrieves the message data. This accepts a TPS keyed list that contains the keys MODE, MSGID, and ARGS, and returns a TPS disposition list.
ibout	A TPS template that wraps the message data into ibmime format. This accepts a TPS keyed list that contains the keys MODE, MSGID, and ARGS, and returns a TPS disposition list.
marp	Used by the <code>hcimsgarchive</code> command to modify the message data to be saved to the external database.
other	Selects any type of Tcl procedure.
routetest	Uses end procedures to process messages produced by the Testing Tool. For example, display message contents, save them to files, or test them for correctness. This is a prototype for end procedures used with <code>hci routetest</code> .
tclalert	The template for custom alerts that are written in Tcl.
tpstest	Uses end procedures to process messages produced by the Testing Tool. For example, display message contents, save them to files, or test them for correctness. This is a prototype for end procedures used with <code>hci tpstest</code> .

Template	Description
tps	Uses a list of Tcl procedures, invoked in sequence.
trxid	Determines a message's transaction ID.
xltCall	Contains a process method that has no return and is given no input values. To do anything, the implementation of this method must use the methods of the XPM object that is passed in.
xltObjects	XLTObjects has an XlateObjects method, taking a vector of object input values and returning a Vector of Object output values. Conversion between system field types and Java classes is performed automatically as specified in the API documentation of the Datum class.
xltString	XLTString has an xlateString method, taking a single string input value and returning a single object output.
xltStrings	XLTStrings has an xlateStrings method, taking a vector of string input values and returning a Vector of Object output values.
X12MetaData	Contains code for all the fields in the ISA and GS segments. These are commented out initially. Uncomment the code in the fields and fill in the appropriate values.
xltp_func_string	A special template of the xltp proc that handles a string within an XLT.
xltp	Receives input through a certain set of variables, and returns data through another set. This proc style only works when called from a code fragment of an XLT.

Text editor

With every new Tcl procedure, **Script Editor** performs three steps before it opens the editor:

- 1 It looks for a Visual environment variable.
- 2 If one does not exist, then it looks for an editor environment variable.
- 3 If one of these variables exists, then the **Script Editor** opens the editor specified within the variable.

If the specified editor does not exist, then:

- UNIX opens vi
- Windows opens Notepad

Script Editor search options

The Script Editor **Search** menu has these options:

Function	Description
Find	<p>This opens the Find dialog box.</p> <p>Specify what to find in the field and select any parameters. Click Find. All matched strings in the editor are highlighted. The right-side vertical bar shows all hints of the search result.</p>
Replace	<p>This is similar to Find. Select what to find from the menu. Select what to replace it with.</p> <p>Click Replace or Replace All to complete.</p>
Go To Line	<p>Goes to the specific code line specified in Line Number.</p> <p>When you click OK, the editor moves the cursor to the specific line of the current <code>tcl</code> file.</p>
Show Find search bar	<p>Opens the Find search bar.</p> <p>This toggles the Find search bar located at the bottom of the Script Editor.</p> <p>This has the same functionalities as the Search dialog box.</p> <p>Note: The Find and Replace search bars cannot be open simultaneously.</p>
Show Replace search bar	<p>Opens the Replace search bar.</p> <p>This toggles the Replace search bar located at the bottom of the Script Editor.</p> <p>This has same function as the Replace dialog box.</p> <p>Note: The Find and Replace search bars cannot be open simultaneously.</p>

Tcl namespace support

In the `tclprocs` directory there is a `.upocindex` file, whose header is:

```
#This index file is used by the GUI
#It is maintained by the mktclindex and should not be edited manually.
#Removing or editing this file will prevent the GUI from displaying
#Information properly.

#Format: { {upoc type} {procname} {filename} }
```

You cannot put procedure names into this file which are meant to be used only as subroutines.

Defining a Tcl namespace

There are two methods of defining a Tcl namespace and its procs.

- Method 1: Define all procs in the namespace eval body:

```
namespace eval testNS {
  proc proc1 { args } {
    #code
  }
}
```

- Method 2: Define procs with a qualified name and outside of the namespace eval body.

```
namespace eval testNS {
}
Proc testNS::proc1 { args } {
  #code
}
```

In the system, only procs that are defined with "Method 2" are saved into the `.upocindex` file and are visible on the IDE's UPoC list. To use namespace in the script, put private procs inside the `namespace eval` body (Method 1). Then, define public procs with the qualified name format (Method 2).

User-defined Tcl templates

User-defined Tcl templates are loaded in sequence from:

- Root level (`$HCIR00T/tclprocs`)
- Master (primary) site level (`$HCIR00T/$MASTERSITENAME/tclprocs`)
- Site level (`$HCIR00T/$SITENAME/tclprocs`)

If duplicate templates exists, then the site level has the highest priority and the root level has the lowest priority.

External Editor and Import Script tool

Use an external editor to create new or open existing Tcl files. Then, import the modified or generated Tcl script files back to the IDE using the **Import Script** tool.

Configuration of the external editor is specified on the **Client Preferences > External Command** tab.

Note: If you do not configure the external editor, then the default editor is used.

To configure an external editor to edit Tcl files that are selected from the Site Manager :

- 1 Specify a `@F` placeholder in the command line.
For example, to edit a file using Windows Notepad, specify:

```
notepad @F
```

- 2 If you use an external editor that does not create its own dialog box when it starts, then you must create a dialog box as part of the command line.

- On Windows clients:

```
cmd /c start vi.exe
```

- On UNIX clients:

```
xterm -e vi
```

- 3 Configure the command line to invoke the external editor with correct parameters.
- 4 When you specify the command line and click **OK**, there is validation to ensure no opened script files are in the built-in editor.
If any opened script files exist, then a message box opens asking you to close all opened scripts.
- 5 After specifying the external editor, you can use that editor to open or create new Tcl files.
To create a new file, launch the external editor from the Launch Bar.
For existing Tcl files, double-click the specified Tcl file from the Site Manager tree.
- 6 At this point, there is another validation to ensure the command is running correctly during runtime when the editor is invoked.
For an existing Tcl file, the file is copied to a temporary directory under %HCIR00T%\integrator\temp\hostname\sitename\ in the IDE with the specified script name.
For a new Tcl file, the editor is directly invoked with the working directory set to be the temporary folder. You must specify an appropriate file name when saving the new file.
If the editor cannot be run correctly, then an error message is shown.
- 7 Click **Import** on the IDE to import the modified or generated Tcl script file to the IDE.
A file browser opens for you to select the imported file path.
If any selected files have been opened and modified with the IDE's built-in editor or are locked, then an error message displays and stops the import.
- 8 The selected files are imported into the current open site.
If there are existing script files with the same name, then a message box opens asking to overwrite it.
- 9 After the import is permitted, all of the actions that the IDE's built-in editor can do are available. That is, importing performs any required transformation, compilation, remote saving, and other tasks that the IDE's built-in editor performs.

Tcl editor use cases

These examples show creating and editing a Tcl procedure.

Creating a Tcl procedure

To use a template in creating a new Tcl procedure that manipulates a message, use the built-in editor to look up command details.

Then, use the matching bracket feature to find the end of a `while` loop.

Editing a Tcl procedure

To edit an existing Tcl procedure, use the search and replace function. This function replaces a keyword with a new keyword in a file.

Site Document

The server-side **Site Documentation** dialog box is used to generate the Site Document.

In Windows, launch the dialog box by:

- Selecting **Start > All Programs > Infor Cloverleaf Integration Suite > Tools > Site Documentation**.
or
- From the command line, specifying `hcisitedocgui`.

In UNIX, launch Site Document from the command line by running `hcisitedocgui`.

You can also access the Site Document through the IDE:

- **Tools > Configuration > Site Documentation**
- **Help > Site Documentation**
- **Launch Bar > Configuration**

Site Document GUI

The IDE is designed to configure a system site, and you review system sites through the IDE. For more flexibility and convenience, you can also remotely review system sites through the Site Document feature.

Site Document shows the available configurations in a specific site using HTML pages. It shows the important site information and configurations, displaying the details, and providing the links to the available configurations.

Configurations are laid out in a formatted fashion for ease in printing.

The Site Document title contains the computer name and system version number. Clicking the title returns to the home page.

Note: Internet Explorer is not supported for displaying the Site Document. You must use MS Edge or Chrome.

The Site Document has these features:

- Includes all editable configurations in the IDE
- Supports all system-supported operations
- Deployable to the web server outside the IDE
- Includes style sheet customization

The Site Document files are located at these locations:

- `hcisitedoc` is located at `$HCIR00T/clgui/bin`.
- `hcisitedocgui` is also located at `$HCIR00T/clgui/bin`.

In the **Site Documentation** dialog box, **Site** lists all selected sites. If the site is a master site, then "(Master)" is appended to the site name.

By default, **Destination Folder** is populated with the value `$HCIR00T/server/tomcat/webapps`. Click **Browse** to open a file browser, where you can change the value to another destination folder.

Server interface

`hcisitedoc` generates HTML pages of the specific site. The command parses all available resource files in the specified site and produces a corresponding set of HTML pages.

```
hcisitedoc -s site_name [site2_name ... [siten_name]
[-d destination_dir] | [-h | -help]
```

- `-s site_name [site2_name ... [siten_name]` specifies for which site the document is generated.
- `-d destination_dir` specifies in which folder the document is generated.
- `-h` or `-help` shows the help usage.

If `destination_dir` is not specified, then the site doc is generated under `$HCIR00T/server/tomcat/webapps/sitedoc`.

After the command line is issued, stdout prints the detailed generating processes. For example:

```
Generating Site Document for Site XXX...
Generating Netconfig...
Generating HL7...
    Generating variant YYY...
    Generating variant ZZZ...
```

Changing the Site Document URL to a different path

In this example configuration, the `sitedoc` is set and open in several locations with different modes.

- 1 Local mode connect to the Host Server.
- 2 Modify the `site_doc_dest_folder` configuration in the site's `siteInfo` file. In this example, `C:\cloverleaf-hs\cis19.1\integrator` is the `HCIR00T`. One of the site's name is `new`.
- 3 Put the `sitedoc` folder in the `C:\cloverleaf-hs\cis19.1\integrator\web` path.

- 4 Configure this path in the `siteInfo` of the new site. The configuration is `site_doc_dest_folder=C:\cloverleaf-hs\cis19.1\integrator\web\sitedoc`.
- 5 Put the `sitedoc` folder in path `E:\folders\site\siteInfo` of the new site. The configuration is and configure this path in the `site_doc_dest_folder=E:\folders\site\sitedoc`.

Example `siteinfo` file:

```
monitord4stats2dbinterval=900
monitord4stats2dbschema=host,site,processes,threads,interthreads
monitord4stats2dbsizecycle=40960
#site_doc_dest_folder=C:\cloverleaf-hs\cis19.1\integrator\server\tomcat\webapps\sitedoc
#site_doc_dest_folder=C:\cloverleaf-hs\cis19.1\integrator\web\sitedoc
site_doc_dest_folder=E:\folders\site\sitedoc
```

- 6 Remote connect to the Host Server. For this, the new configuration is added in `$HCIR00T/server/tomcat/conf/server.xml`.
 - `<Context docBase="../../web/sitedoc" path="/sitedoc" reloadable="true"/>` is added in `server.xml`,
 - `docBase` is the `sitedoc` path.
 - `path="/sitedoc"` is fixed.
- 7 Restart the Host Server. Tomcat starts successfully.
- 8 The other path is `<Context docBase="E:/folders/site/sitedoc" path="/sitedoc" reloadable="true"/>`. This also needs a Host Server restart.

Example `server.xml` file:

```
</Realm>
<Host appBase="webapps" autoDeploy="true" name="hostname" unpackWARs="true"
  <Context docBase="../../docs" path="/cldocs" reloadable="true"/>

  <!--<Context docBase="../../web/sitedoc" path="/sitedoc" reloadable="true"/>-->
  <Context docBase="E:/folders/site/sitedoc" path="/sitedoc" reloadable="true"/>
  <Context docBase="/" path="/" />
  <!-- SingleSignOn value, share authentication between web applications
```

- 9 If Tomcat does not start, then open the Catalina log in `HCIR00T/server/tomcat/logs` and print the error message. Check whether the path is invalid.

Generating and viewing the Site Document

From the host server, you can generate the site doc for any site from the Site Document. Multiple sites can be simultaneously generated.

To generate the site document from the command line, see [Server interface](#).

Generating from the GUI

- 1 On the Site Document GUI, click **Select** to open the a dialog box to specify the sites to generate.

Multiple sites can be selected.

- 2 Click **Apply** to finish the selection.
This populates **Site** with the selected sites.
- 3 Click **Generate** to start generating the HTML files.
This runs `hcisitedoc -s site -d destination`.
Command output is shown in the **Output** text area.
By default, the generated files are put under `$HCIR00T/server/tomcat/webapps/sitedoc`.
- 4 When the generating is completed, click **Done** to finish.
- 5 To open the Site Document, navigate to `$HCIR00T/server/tomcat/webapps/sitedoc` and double-click `index.xml`.

Local and remote access

- 1 Open `server.xml`, located at `$HCIR00T/integrator/server/tomcat/conf`, and verify the listed ports are 15040 and 15047.
If you have a firewall in place, then ensure the port is available. Otherwise, the firewall could prevent the port from being used.
- 2 Open a browser and specify the HTTP address `http://host server:port/sitedoc/index.xml`.
The port is listed on the **Server Administration > Web Server** tab. The default is 15040.
If you are unable to connect using the HTTP address, then use: `https://host server:15047/sitedoc/index.xml`.

Translation Configurator

The Translation Configurator is used to define how incoming records are translated into outgoing records.

The translation thread takes the message, parses the record, and translates it into a new format and often a new protocol. Then it routes it to the destination.

All inbound messages transitioned from inbound to outbound states go through the translation thread. This deals only with engine messages, so there is no protocol driver associated with it.

It also receives and sends all messages by ICL (Interthread Communications Library).

You must know this information to use the Translation Configurator effectively:

- The types of transactions that are processed by each connection. For example, ADT.
- The record layouts for those transactions. For example, HL7, X12.
- The types of data that are handled by each transaction. For example, flat record.
- The way that data is processed or translated. For example, FTP.

This information is pre-defined by your institution's business procedures.

Translation pipeline

The running of a given translation action can be thought of as a pipeline.

At each stage, error checking is performed.

In case of an error, the error control value determines the result.

After an error condition is handled, any remaining parts of the pipeline are cleared. The translation continues with the next translation line, unless the **Error** value is "Error."

The stages of running, along with the potential error conditions at each stage, are:

- Data Retrieval: Data values not available and no default given.
- Pre-Processor Code Segment Execution: Tcl code errors out for any reason.
- Main Translation:
 - CALL: Tcl code errors out for any reason.
 - COPY: No potential errors.
 - MATH operations: Non-numeric input, math error.
 - TABLE: Named table, or side, not found.
- Post-Processor Code Segment Execution: Tcl code errors out for any reason.

Translation configuration

Use the Translation Configurator to define how incoming records are translated into outgoing records.

Formats are selected from the **Choose File Formats** dialog box (**File > New**). The remainder of the pane could change, depending upon the chosen format.

- For EDIFACT, HL7, HPRIM, LDL, NCPDP Telecom, NCPDP SCRIPT, NCPDP Formulary and Benefit, and X12, select the **Version**, **Variant**, and **Message**.
- For FRL, VRL, HRL, or XML, select the **File**.
- For JSON, select the **Package**. The root, master (primary) site, and current site JSON packages are shown on the list. The **Package** selection determines the JRL file list.

The root-level JSON package name is *fhir*, and is located at `$HCIRoot/formats/json`.

- For Database Schema, select a **Connection** and **Table Schema**.
- For XML, select the tested XML file. The files on the menu are the tested DTD/Schema/DTD-containing-XML files. Tested files are located in package folders managed by the XML Package Manager. They are listed on the menu using the form: *Packagefolder\root_DTD_etc_name*.

The list is empty if no DTDs are configured. Files are tested using the XML Package Manager.

This test must be performed before opening the Translation Configurator's **Choose File Formats** dialog box.

For **Insert an Existing XLT**, click the arrow to insert an existing translation file.

Note: The tree views on the IDE main panel are only for viewing the message structure. They cannot be used to specify message paths into actions. Use the tree views on the Action Properties pane to create message paths in actions.

Translating incoming records into outgoing records

- 1 Select **New** to open the **Choose File Formats** dialog box, where you configure the Input and Output formats.
To reconfigure an existing configuration, select **File > Reconfigure**.
- 2 Select **New Action Mode** and then select the action. Choices are Append, After, or Before.
- 3 Specify the source and destination values using the Input Message Format and Output Message Format panels. These panels open on the side of the Action Properties pane or operation list.
- 4 Test the Configuration. Select **File > Test** to use the Testing Tool.
incoming and outgoing record file formats must be predefined in the configurator for each type. These predefined formats are saved with an extension denoting its type. For example, `.frl`. This is the file to which the **Choose File Formats** dialog box refers.

Parsing and translation

The Translation Configurator uses parsing and translation for a wide variety of data formats and translation actions.

The Translation Configurator supports these constructs:

- Many fields to many fields
- Many fields to one field
- One field to many fields

The pre-processor functions prepare the data for translation. The post-processor functions work on the message after translation.

Saved formats

When creating a new translation configuration using the last selected format, ensure the **Remember the last selected formats** option in **Client Preferences > Xlate Config** tab is selected.

This is saved in `client.ini` as `xlt_remember_last_format`.

When this option is selected, the last selected input and output record formats are saved in `clide.ini`. Formats are saved with the keys `xlt.inbound.format` and `xlt.outbound.format` when **OK** is clicked on the **Choose File Formats** dialog box.

For example, `xlt.inbound.format=hl7 2.7 ADT_A01`. Subsequent translation configurations have the saved values already selected.

LDL formats

The fixed index number for fields has been removed for GRM addresses. Because of this, the **Index notation for field addresses. . .** option in **Options > Client Preferences > Xlate Config** does not work for LDL formats in Translation Configurator. The GRM address for LDL is not changed whether the option is selected or cleared.

Saving the last format

Multiple interfaces are required, all dealing with the same data formats.

- 1 In the IDE, go to the **Options > Client Preferences > Xlate Config** tab.
- 2 Select **Remember the last selected formats** to enable the Translation Configurator to remember the last selected data formats.
- 3 In this example, create a translation called Translation1 with these parameters:
Input format:
 - HL7 2.6
 - variant Cerner
 - message ADT_A01Output format:
 - HL7 2.7
 - variant Epic
 - message ADT_A01
- 4 Save the translation. You can now create another translation that uses the input and output formats from Translation1.

Creating an action

After selecting the file formats, create a new action.

- 1 Select the New Action from the toolbar or from the right-click menu in the layout panel. The mode determines where the action is placed.
 - **Append** places the action at the end of the currently selected branch, as the last child of the currently selected action's parent..
 - **Before** places the action before the current selection on the Translation Configurator. The action is inserted relative to the currently selected action's parent.
 - **After** places the action after the current selection on the Translation Configurator. The action is inserted relative to the currently selected action's parent.
- 2 Select the action by clicking the appropriate tool on the New Action toolbar. This places a new empty action at the location. Action choices are:
 - **COPY** copies the source values directly to the destination fields.
 - **CONCAT** concatenates two or more input fields into a single output field.
 - **TABLE** translates the source values according to the specified translation lookup table.

- **ITERATE** defines an iterative block of translation actions. The block repeats every time the specified basis is encountered in the input record. The basis Type can be a record variant Field, Group, List, or Segment.

When Type is user, the Basis can be a user-defined variable that stores a list.

- **MATH** performs the specified math operation on the source values. Specify at least two source values for a math operation. One or more can be a constant value.
 - **ADD** adds the source values.
 - **AVG** takes the average of the source values.
 - **DIV** divides the source values.
 - **MAX** takes the maximum of the source values.
 - **MIN** takes the minimum of the source values.
 - **MOD** takes the remainder after dividing the source values.
 - **MUL** multiplies the source values.
 - **SUB** subtracts the source values.
- **STRING** provides string operations, such as Trim, Substring, Concat, and others.
- **COMMENT** includes a text comment with the translation definition.
- **BULKCOPY** copies every common input field to output.
- **PATHCOPY** copies, clears, and replicates information that is grouped at almost any level of association.
- **CALL** calls a Tcl procedure to perform a set of operations on the source values.
- **CONTINUE** generates a message from all the translated elements up to the Continue, and places it on the disposition list as CONTINUE. This includes further processing of the generated message.
- **DATECOPYOPT** defines how to handle date copies.

IF provides controls for defining a condition and the action to be taken if that condition exists.
- **SEND** generates a message from all the translated elements up to the send, and transmits that message to the outbound connection. This action permits the generation of multiple outbound messages from a single inbound message. The SEND and CONTINUE order is important. This bypasses normal processing and sends the generated message to the Partial Queue. Otherwise, it is the same as CONTINUE.
- **SUPPRESS** transmits the message. This action permits the generation of multiple outbound messages from a single inbound message. It suppresses processing of the original input message beyond the translation phase. SUPPRESS is the last translation operation if SEND or CONTINUE operations generate multiple outbound messages from one inbound message.
- **BREAK** is used to break a loop or to skip iterates.
- **INCLUDE** can use a reference in the xlate, where you create one xlate for the PID translation and refer to it in the other xlates.

- 3 After selecting the action, edit the details of the operation by clicking the action and editing the Source/Destination values, adding procs, and so on.

Address indicator node

Cloverleaf can handle both `address type entry(0).resource.resourceType` and `entry(0).resource.^Account.resourceType`.

- `entry(0).resource.resourceType` is the standard address type.
- `entry(0).resource.^Account.resourceType` uses the caret symbol (^) as an indicator node.

The indicator node processes the name conflict issues. For example, FHIR is a special instance of JSON, that has many `anyOf/allOf` nodes in the definition.

For those nodes, Cloverleaf uses the first node that has the same name. In FHIR, some nodes might have different definitions, even though they have same name.

In this case, the message might not be what the user needs or is expecting. When this happens, you can use the caret symbol (^) as an indicator node.

For the translation `INCLUDE` action, when there is a name conflict issue, an indicator node is required in the sub `xlate` file.

Example:

In the Translation Configurator:

- 1 Configure the JSON format for inbound or outbound.
- 2 Select the sub-node of the `anyOf/oneOf/allOf` node.
- 3 Add a ^ ("caret" symbol) prefix to the original address.

In another example:

`resource` is a `oneOf` node. The sub-node is `reference`.

The address changes from `entry(0).resource.resourceType` to `entry(0).resource.^Account.resourceType`.

If the sub-node is not referenced, then the address is changed from `entry(0).resource.resourceType` to `entry(0).resource.^(0).resourceType`.

`^Account` and `^(0)` are the added nodes.

The ^ ("caret" symbol) is the indicator.

XML and the Translation Configurator

The Translation Configurator employs many XML features, some of which are:

- Using an XML tree structure, where the input and output format lists use trees to show message structure.
- Placing XML messages into hierarchical division of message types.
- Using an XML schema instance with the option to re-declare an element type to something other than the stated type within the schema definition.
- Supporting XML schema wildcards.
- Using XML path strings, where you can state at a particular path component that subsequent components in the path are parsed/validated.

XML tree structure

The input and output format lists use trees to show message structure. Although details may differ from format to format, all trees fall into two categories:

- Simple message formats.
These organize data into trees that maintain fixed parent/child relationships. That is, if a child is shown in the tree, it displays in the message. These formats tend to have only a few permissible generational levels of parents begetting children, who beget children, and so on.
- Hierarchical message formats.
These organize data in a more complex manner. Generational depth is not necessarily restricted. Portions of a message displaying in a tree may be optional, and portions of a tree may repeat in the actual message. This type of tree is called HMD, or Hierarchical Message Data. One such HMD format is HL7. Optional portions of the tree are indicated by bracket pairs ([]). Potentially repeatable portions are indicated by brace pairs ({ }).
HMD respects definitions based on the priority of local site, master (primary) site, and root.

XML messages

XML messages fall into the hierarchical division of message types. The XML DOM (Document Object Model) considers the elements of XML documents to fundamentally exist in parent/child tree relationships. The full W3C recommendation for XML includes new organizational concepts, including:

- Conditionality, where an XML message element may exist in a relationship with another contender element for the same location in the message.
- Infinite recursive constructs, where a message element can refer to itself.

XML also recognizes a fixed, sequential organization in which one message element must follow another message element within the message. It also generalizes the concepts of optionality and repeatability under the heading of cardinality.

XML symbols

HMD trees express cardinality concepts using { }, [], and [[]]. These symbols do not map well to the XML cardinality designators ?, *, and +.

These symbols are used in XML trees:

- ?
This indicates "zero or one instance."
- *
This indicates "zero or more instances."
- +
This indicates "one or more instances."
- ,

This indicates "must follow."

- |

This indicates "either/or."

The XML tree uses "anonymous group" nodes that contain and express these relationships between child nodes.

XML tree view labels

On the tree view, minimum and maximum instances are shown in "min: minOccurs, max: maxOccurs." Sequence and choice are shown in "Seq." and "Choice."

When maxOccurs is defined as unbounded in the XML schema, the tree view shows "max: unb."

The attributes property dialog box opens when you right-click a node and select **All Schema Attributes** from the menu. This shows the value of the XML schema attributes. This dialog box shows several attributes according to the node type.

Tree icons

This table shows the data types of all the format types. Data types that are shown in parentheses indicate they use the same icon as the prior types.

Format type	Data type
XML	Content (Subfield), Declaration, Attribute, Element, Group
FRL	Subfield, Field, Group
VRL	Subfield, Field, Group
HRL	Subfield, Field, Segment, Group
HL7	Subfield, Field, Segment, Group
EDIFACT	Subfield, Data Element (Field), Composite Data Structure, Data Segment (Segment), Group
X12	Subfield, Data Element (Field), Composite Data Structure, Data Segment (Segment), Group
NCPDP Telecom	Subfield, Field, Composite Data Structure, Token (Segment), Group
NCPDP SCRIPT	Subfield, Data Element (Field), Composite Data Structure, Data Segment (Segment), Group
NCPDP F&B	Subfield, Data Element (Field), Data Segment (Segment), Group

Show and hide

Right-click a node and select **Set Root** from the menu. This sets the selected node as the root node of the tree view and to hide the other views.

Click the "up" arrow at the bottom of the message format pane to show the parent and sibling nodes of the current root.

Right-click the tree view and select **Reset** from the menu to show the original Inbound or Outbound message format tree.

Right-click the tree view and select **Up Level** from the menu to show the parent and sibling nodes of the current root.

Showing all fields and subfields in XML format

For the XML format, right-click a node and select **Show > All Fields** to show all the fields of all the loaded nodes.

Select **Show > All Subfields** to show all the subfields of all the loaded nodes.

XML hierarchy

After the Input or Output Message Format panel is opened, XML message formats display as a hierarchical tree, similar to the other message formats. XML does not employ rigid message definitions.

XML messages have the potential to be infinitely deep in hierarchy. With the XML specification, you can use concepts such as "recursion" and introduces more detail to repeatability. As a consequence, the tree structure that is shown in the dialog box is often more complex than with other message formats.

This shows the representation of the DTD:

```
<!ELEMENT alpha2 ((gamma,foo)+,(delta,foo)*)+>
<!ATTLIST alpha2
    one CDATA#IMPLIED>
<!ELEMENT beta EMPTY>
<!ELEMENT gamma (delta,foo)+>
<!ELEMENT delta (#PCDATA|beta|foo)*>
<!ELEMENT foo (#PCDATA)>
```

The names that are shown on the tree nodes correspond to XML elements or attributes. Tree lines, indentations, and node icons represent the hierarchy of the elements.

Element repeatability is represented by the syntax (#,#). For example, (1,1) is "one and only one" and (1,-1) is "one or more."

Element sequencing is represented by the symbol (_,), where "_" can be occupied by one of:

- Comma (,), which represents the concept of `list`.
- Pipe (|), which represents the concept of `or`.

Attributes are designated with an ampersand (&).

Recursion

XML permits recursion, where a message definition can contain self-referential structures.

In this example, the recursive node is `gamma`. The recursion is indicated through visual clues:

- The plus sign (+) indicates the node is unexpanded.
- The element name is not followed by any repetition or sequence information.

In this example, it is expanded twice. You can expand the tree infinitely, until the client machine's resources are exhausted. Consequently, this feature must be used with care. The expansion in this diagram provides user-access to the `#text` element contained within the third instance of `gamma`. Upon selection of a particular node, the path string is generated.

XML path string generation

As with other message types, nodes in the XML message structure tree relate the structure. Leaves are locations that are occupied by message data. For example, a translation action dialog box is opened and a tree item is selected that corresponds to a datum location. Then, the input/output argument widgets are enabled on the structure dialog box. You can then select the appropriate widget to set a path string corresponding to the datum location as an argument to the open translation action.

You can also specify the path specification manually into the translation action dialog box, in the same manner as with other message types.

This shows the treatment of repeatability with the XML path strings. The selected item is identified by and generates the basic path:

```
alpha2.0(0).0(0).gamma.0(0).delta.0(99).foo.#text
```

The "(0)" portions of the example path string are shown in the path presentation text box at the bottom of the dialog box. These portions of the string represent nodes that can repeat. They are small entry boxes where you can type the repetition values.

You are responsible for entering correct repetition values. The XML Schema syntax is capable of expressing upper bounds to permissible repetitions within an XML message structure.

The path string repetition dialog boxes use "(0)" as the default value.

Attribute "use" support

The `use` property in the attribute definition declares the attribute to be `required`, `optional`, or `prohibited`.

For example, in this schema definition:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="myElement">
    <xsd:complexType>
      <xsd:attribute name="a" use="required"/>
      <xsd:attribute name="b" use="optional"/>
      <xsd:attribute name="c" use="prohibited"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```
</xsd:element>
</xsd:schema>
```

These instances are both permitted within this definition:

```
<myElement a="mya"/>
<myElement a="mya" b="myb"/>
```

This is not permitted:

```
<myElement a="mya" c="myc"/>
```

XML message encoding

For fields that are required in the schema, the Translation Configurator creates an empty element. An example of a required field is `nullable=true`. It does this in case there is no COPY, or other action to populate it. An empty element is also created if there is a COPY, or other, action that does not put any value in the element.

The element must be there, for the schema to be valid. When this happens, the Translation Configurator does not create an XML that does not validate against its schema.

When encoding an XML message, the schema is checked for elements and attributes that are required but not populated during translation. If found, then the engine automatically inserts an empty element or empty attribute in the XML message.

For elements, an element with no value is inserted if the element is a `simpleType` or if its `nullable` flag is true; otherwise, an empty structure is inserted.

For attributes, an attribute with no value is inserted.

Example

The schema is defined as:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="test">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="RequiredElement1" minOccurs="2"
type="xs:string" maxOccurs="unbounded">
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

Only the first repetition is populated:

```
COPY =ABC -> test.RequiredElement1(0).#text
```

The output is:

```
<test><RequiredElement1>abc</RequiredElement1><RequiredElement1/></test>
```

In this example, the empty element is automatically inserted by the engine.

Exceptions

There are a few exceptions in which the automatic insertion does not happen:

- Abstract elements are not automatically inserted.
- Elements whose type are abstract are not automatically inserted.
- An ANY element is not automatically inserted.
- For recursive elements, automatic insertion stops at the first repeated element encountered.
- Only the first element displaying in the ALL groups are inserted. The rest are not automatically inserted.
- Data type validations are not checked when the empty element or attributes are inserted. For example, if empty strings cannot be used in an attribute's data type, then an attribute with no value is inserted if the attribute is found missing.

Note: The inserted empty elements or attributes might cause the resulting XML to be invalid. You are responsible for making them valid by populating the correct values in translation.

XML type substitution

In an XML schema instance there is an option to re-declare an element type to something other than the stated type within the schema definition. This is indicated by an additional `xsi:type` attribute in the XML instance whose value indicates the name of the type to validate the element against.

This type must be a globally defined type in the schema definition. It must also be the type that is defined or a type that is derived from the defined type.

See <http://www.w3.org/TR/xmlschema-0/#UseDerivInInstDocs>.

This offers a detailed description of type substitution.

For example, having defined an element as `myElement` to be type `base_type`, and type `base_type` has a derived type `derived_type`:

```
<xsd:element name="myElement" type="base_type"/>
<xsd:complexType name="base_type">
  <xsd:sequence>
    <xsd:element name="myChild1" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="derived_type">
  <xsd:complexContent>
    <xsd:extension base="base_type">
      <xsd:sequence>
        <xsd:element name="myChild2" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

This XML instance would be valid:

```
<myElement><myChild1/></myElement>
```

A re-declaration of the defined type is also valid:

```
<myElement xsi:type="base_type"><myChild1/></myElement>
```

To re-declare the `myElement` type from `base_type` to `derived_type`, you must have the child elements for type `derived_type`. You must also have the `xsi:type` attribute in your XML instance. For example:

```
<myElement xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="derived_type">
  <myChild1/>
  <myChild2/>
</myElement>
```

Configuring a successful XML substitution

The XML substitute function works correctly when it is on the translation's source side. However, when the XML substitute function is used on the translation's destination side, an "Invalid path error" is given.

For example:

- 1 Use the `substitute` function on the xlate destination side. In the xlate file **HL7 > XML**:

```
{ { OP COPY }
{ ERR 0 }
{ IN {{0(0).MSH(0).#3(0).[0]}} }
{ OUT topElem.customer{kundeinfoType}.name.#text }
}
{ { OP COPY }
{ ERR 0 }
{ IN {{0(0).MSH(0).#3(0).[0]}} }
{ OUT topElem.customer{kundeinfoType}.&gender }
}
test data:
MSH|^~\&|HIS|||20090603090227|ADT^A03|20090603090227|P|2.1||| EVN|A03|20090603090227|||
PID|||200062||
Irving^Bethany^Jo||19910125|F||2106-3|1290 RIVERSIDE DR^^IRVING^TX^75039|439|||EN
GLISH|S|SPI|3000014|123335566||
PV1||I||A|||10091|10091|00000^00000^00000^00000|MED|||7|||10091^Einstein^Al
bert^^Dr.|IE|1|CI|||||||
AA|||||200905110906|200906030859|3121800|3121800|||
```

When you use `hcixlttest`, there is an invalid path:

```
[0:TEST] Invalid path: topElem.customer{kundeinfoType}.name.#text - Type substitution
{kundeinfoType}
was not found in instance.
```

- 2 Now use `substitute` on the xlate source side.

```
In xlate file xml--->hl7
{ { OP COPY }
{ ERR 0 }
{ IN topElem.customer{kundeinfoType}.name.#text }
{ OUT {{0(0).PID(0).#5(0).[0]}} }
}
{ { OP COPY }
{ ERR 0 }
{ IN topElem.customer{kundeinfoType}.0.kundeitem.#text }
{ OUT {{0(0).PID(0).#5(0).[1]}} }
}
test data:
<?xml version="1.0" encoding="UTF-8"?>
<topElem>
  <customer xsi:type="kundeinfoType" gender="male" >
```



```
<name>John Smith navn</name>
<kundeitem>hhhkunde</kundeitem>
</customer>
</topElem>
```

There is now no error when running `hcixlttest`. The XML substitute function works correctly.

Abstract types

Declaring a type as abstract requires the use of a type derived from it, and identified by the `xsi:type` attribute, in the instance document.

After the IDE determines an element's type is abstract, the element should not be expanded until it is substituted. The element whose type is abstract is rendered in italic type. You can right-click the element node and substitute its type. Then, address an address pointing to the substituted type. The substituted element does not show the abstract icon because only non-abstract types can substitute abstract elements.

Declaring a type as abstract

In the schema definition, a type can be declared `abstract`.

```
<xsd:complexType name="xxx" abstract="true">...
```

This declaration means that elements that are declared with this type must have their types re-declared in an XML instance. To do this, use an `xsi:type` attribute to declare a non-abstract type derived from this abstract type. Elements that are declared with this type that do not have an `xsi:type` attribute are incorrect.

For example, using the previous example but making `base_type` an abstract type creates:

```
<xsd:element name="myElement" type="base_type"/>
<xsd:complexType name="base_type" abstract="true">
  <xsd:sequence>
    <xsd:element name="myChild1" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="derived_type">
  <xsd:complexContent>
    <xsd:extension base="base_type">
      <xsd:sequence>
        <xsd:element name="myChild2" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

These XML instances would no longer conform:

```
<myElement><myChild1/></myElement>
<myElement xsi:type="base_type"><myChild1/></myElement>
```

In this case, a non-abstract type must be declared. This instance would still conform:

```
<myElement xsi:type="derived_type">
  <myChild1/>
```

```
<myChild2/>
</myElement>
```

This is a schema element declaration for compile:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified" at
tributeFormDefault="unqualified">
  <xsd:element name="myElement" type="base_type"/>
  <xsd:complexType name="base_type" abstract="true">
    <xsd:sequence>
      <xsd:element name="myChild1" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="derived_type">
    <xsd:complexContent>
      <xsd:extension base="base_type">
        <xsd:sequence>
          <xsd:element name="myChild2" type="xsd:string"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>
```

XLT substitution action

- 1 In the Translation Configurator's Action panel, right-click an element node in the XML Input Message Format pane or XML Output Message Format pane.
- 2 Select the **Substitute** option from the menu. This opens the **Substitutable Type Selection** dialog box. If the element node's type does not have derived types, then this option is not available. The format tree shows the attribute as `&attributeName {min:xx,max:xx,1} (type)`.
- 3 From the **Substitutable Type Selection** dialog box, select from a list of substitutable types. After you click **OK**, the element node is replaced with the selected type.

For an outbound substitution, in addition to setting the substituted type, set the `xsi:type` attribute. This must be set so that the type attribute is populated with the substituted type declaration.

Substitutable Type Selection dialog box

The **Substitutable Type Selection** dialog box lists all the substitutable non-abstract types, including the derived types and the base type itself that is not abstract.

For example: Type B is an extension from type A. Type C is an extension from type B.

When a type A node is selected, the dialog box lists A, B and C.

- If A is abstract, then B and C are listed.
- If both A and B are abstract, then only C is listed.
- If all of them are abstract, then no type is listed and **OK** is not available.

When you click **OK**, the sub-tree of the selected element node is replaced with the chosen type.

After the tree structure is changed, you can get the correct addressing to match the instance document. The changed tree structure is not saved for the next open action.

XML schema wildcard support

Note: All of the schema built-in types are supported for substitution for `xsd:anyType` and `anySimpleType`. OCM's that are compiled from 5.4.1 continue to work as before, without `anyType` support. Only those schemas that are compiled in 5.6 or later can use type substitution for types defined with `anyType`. Access to `##any` and `##anyAttribute` is only available for XML definitions that are compiled in 5.6 and later.

In XML Schema there are structures for targeted flexibility in content models and attribute declarations. This wildcard XML content was not handled fully by versions before 5.6. This content was not addressable within an XLT because there was no deterministic definition of what form this content may take.

There are four XML Schema structures that can have wildcard content:

- `?any?` schema tag
See <http://www.w3.org/TR/xmlschema-1/#Wildcards>.
This tag permits any XML element node content as long as ambiguity constraints are still satisfied. There is also an optional namespace constraint that can be specified in the `?namespace?` attribute.
- `?xsd:anyAttribute?` schema tag
See <http://www.w3.org/TR/xmlschema-1/#declare-type>.
This tag permits any XML attribute node content as long as ambiguity constraints are satisfied. Similar to the `?any?` tag, there is also an optional namespace constraint.
- `?anyType?` built-in schema type
See <http://www.w3.org/TR/xmlschema-1/#d0e9252>.
With this tag, the element that is defined using this type can use any XML element node or XML attribute node content for this node. This `?anyType?` is also the root of all defined types.
- `?anySimpleType?` built-in schema type
See <http://www.w3.org/TR/xmlschema-1/#d0e16395>.
This is the root for all simple types. Any text content is permitted. Node content is not permitted.

User interface

For substitutions, if `anyType` is the defined type for an element, every other built-in and user-defined type in the schema are valid for this type's substitution.

As there are over forty built-in types, only the user-defined types are included in the GUI's list of substitutions.

To use a built-in schema type to substitute (for example, `xsd:string`), you must specify that type by hand.

Note: The substitution of built-in types using the GUI is not supported.

User-defined types that have no base type or whose base type is a built-in type have `xsd:xsd:anyType` or `xsd:anySimpleType` as their base type. This is because the only built-in type that may be substituted is `xsd:anyType` and `xsd:anySimpleType`.

All other built-in types are blocked from substitution by default.

Examples of user interface representations of wildcard contents

For any, an element is defined by a complex type containing any, such as:

```
<xs:element name="childAny">
  <xs:complexType>
    <xs:sequence>
      <xs:any/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

If a namespace constraint is specified, then its value is annotated with any of these possibilities:

- `##any (##local)`
- `##any (uri1 uri2 uri3 list_of_uris)`
- `##any (##other)`
- `##any (##targetNamespace list_of_uris)`
- `##any (##other <targetNS>)`

(if targetNamespace is specified in the schema.)

- `##any (##targetNamespace <targetNS> list_of_uris)`

(if targetNamespace is specified in the schema.)

For anyAttribute, an element is defined using a type containing xsd:anyAttribute, such as:

```
<xs:element name="childAnyAttribute">
  <xs:complexType>
    <xs:sequence>
      .
      .
      .
    </xs:sequence>
    <xs:anyAttribute/>
  </xs:complexType>
</xs:element>
```

If a namespace constraint is specified, then its value is annotated with any of these possibilities:

- `##anyAttribute (##local)`
 - `## anyAttribute (uri1 uri2 uri3)`
 - `## anyAttribute (##other)`
 - `## anyAttribute (##targetNamespace list_of_uris)`
 - `## anyAttribute (##other <targetNS>)`
- (if targetNamespace is specified in the schema.)
- `## anyAttribute (##targetNamespace <targetNS> list_of_uris)`

(if targetNamespace is specified in the schema.)

For anyType, an element (for example, `childxsd:anyType`) is defined using `anyType`, such as:

```
<xsd:element name="childAnyType" type="xsd:anyType"/>
```

Its content is represented as including an optional unbounded sequence of `any` and `xsd:anyAttribute`.

For anySimpleType, an element is defined using `anySimpleType`, such as:

```
<xsd:element name="childAnySimpleType" type="xsd:anySimpleType"/>
```

XML support for DTD ANY and schema anyType

DTD and schema permit elements to be defined to have any content. These definitions look similar to:

- DTD:

```
<ELEMENT foo ANY>
```

- Schema:

```
<xsd:element name="foo" type="xsd:anyType" />
```

```
<xsd:element name="foo" />
```

The element `foo` can have any child elements or attributes or text as its content without explicitly specifying them.

This is supported by treating these declarations as if they were strings. The OCM is created as if the declarations were written similar to these:

- DTD:

```
<ELEMENT foo (#PCDATA)>
```

- Schema:

```
<xsd:element name="foo" type="xsd:string" />
```

XML path semantics

On the XML path string, you can state at a particular path component that subsequent components in the path are parsed/validated. This assumes the modified element type has been substituted by another type using an `xsi:type` attribute at runtime.

To indicate a type modifier on an XML path component, the XML path uses an illegal character in XML element names to prevent character collision.

The "{" character indicates the text is the name of the substituted type, ending at the "}" character. XML path components that are specified after the re-declared component walks the hierarchy based on the substituted type.

These path examples use this schema for validation:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="root">
    <xsd:element ref="child" maxOccurs="unbounded"/>
  </xsd:element>
  <xsd:element name="child" type="abstracttype"/>
  <xsd:complexType name="abstracttype" abstract="true">
    <xsd:attribute name="abstractattr"/>
  </xsd:complexType>
  <xsd:complexType name="realtype">
    <xsd:simpleContent>
      <xsd:extension base="abstracttype">
        <xsd:attribute name="realattr"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
  <xsd:complexType name="otherrealtype">
    <xsd:complexContent>
      <xsd:extension base="abstracttype">
        <xsd:sequence>
          <xsd:element name="othergrandchild" type="xsd:string"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>
```

Path to fetch/store `xsi:type` value

If there is no type modifier defined, then the path is parsed and validated against the declared type in the schema.

For example, in the XML instance:

```
<root>
  <child xsi:type="realtype" realattr="foo"/>
</root>
```

This address represents the value of the `xsi:type` attribute:

```
root.child(0).&xsi:type
```

Because `xsi:type` is a valid child for this element and is not tied to a particular type, this path is correct. The value returned is `realtype`.

- In this case, where `xsi:type` is the source value (a retrieve), it validates whether the source value has a substituted type. If the value is not found, then this results in a `FETCH_FAILED` error.
- If the `xsi:type` is the target value (a store) or if there were already children or attributes added, then this returns an error. This is because the type cannot be re-set until the element has been cleared using `PATHCOPY`.

Type modifier must be used for all attributes and children of type-substituted element.

Based on the previous section's example, this path is not permitted:

```
root.child(0).&realattr
```

Because `realattr` is only permitted when `child` has been re-typed as `realtype`, when this path is evaluated against the instance there is an error:

XML Path Resolution Error: `&realattr` is not a valid attribute of element `child(0)` of type `abstracttype`.

Source type modifier must match XML instance

For an example, an XML instance document contains this content:

```
<root>
  <child xsi:type="realtype" realattr="foo"/>
</root>
```

The XML path that is used to select the value of `myrealtypeattr` is:

```
root.child{realtype}.&realattr --> @attrValue
```

In this example, if the specified type is not correct, then this is also not correct:

```
root.child{otherrealtype}.&realattr --> @attrValue
```

Because the instance has a defined child's type as `realtype`, this type modifier is incorrect and produces this error:

XML Path Resolution Error: The specified element `child` of type `realtype` does not match path type modifier `otherrealtype`.

Target xsi:type must be set before descendent elements or attributes are stored

When the XML is selected for the target message, the `xsi:type` attribute must be set before referring to the sub-element with a type modifier. For example:

```
=realtype --> root.child.&xsi:type
=myvalue --> root.child{realtype}.&realattr
```

In this example, if the type has not been set, then it is incorrect:

```
=myvalue --> root.child{realtype}.&realattr
```

Because the child's defined type does not have the `realattr` attribute, it produces this error:

XML Path Resolution Error: The specified element child of type `abstracttype` does not match path type modifier `realtype`.

Target xsi:type cannot be changed after it is set and children have been added

If the type has been set and then reset, as in this example, it is incorrect:

```
=realtype --> root.child.&xsi:type
=myvalue --> root.child{realtype}.&realattr
=otherrealtype --> root.child.&xsi:type
```

Because the child's type has already been set to `realtype`, resetting it to `otherrealtype` is not permitted and produces this error:

"XML Path Resolution Error: The substituted type of element child has already been set to `realtype` and cannot be changed to `otherrealtype`. Use `PATHCOPY` to copy @null to this element and its descendants."

The only way to change an element's type after it has been set is to `pathcopy @null` to that element. The element itself is then deleted along with its descendants.

Path with type modifier and repetition modifier

If a component of the path is defined with a repetition, then the meaning of the repetition modifier is unchanged.

For an example, an XML instance document contains this content:

```
<root>
  <child xsi:type="realtype" realattr="foo"/>
  <child xsi:type="otherrealtype">
    <grandchild>mytext</grandchild>
  </child>
  <child xsi:type="realtype" realattr="bar"/>
</root>
```


The XML path that is used to select the value of `realattr` in the third repetition of `child` would be:

```
root.child(2){realtype}.&realattr
```

In this example, the `(2)` modifier still indicates the third element named `child`. For the remainder of the address, assume that this third repetition of the element has been redefined to be type `realtype`. If no repetition is indicated, then the default repetition is `0`.

If you type in the address manually and reverse the order of these modifiers, then it does not change the order of operation or the interpretation.

Message flow between threads

Messages arrive by the inbound side of the protocol thread. From there, they are routed and translated through the translation thread before departing through the outbound side of the protocol thread.

A FIFO (First In First Out) sequence is used within a given message priority. Messages that are placed into a queue with the same priority are removed from the queue in the same order: oldest first.

Source and destination values

Temporary variables are accessed by right-clicking within the relevant text entry components (Source, Destination, Basis) and selecting a variable from the menu.

After an action is selected, on the Input/Output Message Format panels, specify the source and destination values for each translation.

Clicking **Toggle Input Format** or **Toggle Output Format** opens the tree view on the side of the dialog box. You can double-click a path or use **Add to Source** or **Add to Destination** below the tree view to insert a path into the action.

Use the input and output lists to identify and select references to the location of particular data items in an input or output message. Selecting an item produces a path string that encodes the selected datum location.

This path string is used as an argument to guide a translation action.

- The path string for an input datum is used to bring that datum into the translation action.
- The path string for an output datum is used to place the output of the action into the output message.

Source and destination value selection

Select source values from:

- Input record components
- Output record components. These display beginning with "~."
- Constant literal values. These display beginning with "=".

- Temporary fields. These display beginning with "@").

A source value is required; otherwise, the corresponding operation is skipped. If you must input empty source data, then you can set "=", without quotes, into the **Source** field.

Select destination values from output record components or temporary fields (these display beginning with @).

JSON auto-expansion levels

The JSON format supports auto-expansion levels in the Translation Configurator.

The default value is "5". The maximum value is "99".

For details, see the JSON auto-expansion level option description at [Xlate Config options](#) on page 80.

Hardcoding literal values

Populate the Source pane of the Action Properties pane as you would Tcl. Use the equal sign to hard-code a literal value into the destination value.

Escape characters must be used for certain sequences. For example, to add a forward slash to a field, escape it with a back slash. The code would then look similar to =\/.

Selecting fields from Input Message Format panel

The Input Message Format panel lists all fields and subfields for the chosen input record format. To make a selection from this list box:

- 1 Click **Toggle Input Format** on the Action Properties toolbar. This opens the Input Message Format panel. The fields and subfields for the chosen format display in the list box.
- 2 Click the plus sign (+) next to the field name to view its subfields.
- 3 Select a single subfield or shift-click to select a group of subfields. The subfield or range of subfields in the path is shown at the bottom of the Input Message Format panel.
- 4 Double-click the selected field/subfield to add it to the source list or right-click in a blank area of the list to open a menu.
Click the appropriate button at the bottom of the dialog box to select where to place the selected field or subfield:
 - **Add to Source.** This places the value in the Source list box of the Action Properties pane.
 - **Copy to Basis.** This is used for ITERATE or IF/THEN/ELSE.
- 5 Close the Input Message Format panel. If the Input Message Format panel is closed, then the next time it is opened, it is restored to its last location.
- 6 If there are no other input or output values to add, then click **OK**. Otherwise, specify output values.

Selecting fields from Output Message Format panel

The Output Message Format panel lists all fields and subfields for the chosen output record format. To make a selection from this list box:

- 1 Click **Toggle Output Format** on the Action Properties toolbar. This opens the Output Message Format panel. The fields and subfields for the chosen format display in the list box.
- 2 Click the plus sign next to the field name to view its subfields.
- 3 Select a single subfield or shift-click to select a group of subfields. The subfield or range of subfields in the path is shown at the bottom of the Output Message Format panel.
- 4 Double-click the selected field/subfield to add it to the destination list or right-click in a blank area of the list to open a menu.
Click the appropriate button at the bottom of the dialog box to select where to place the selected field or subfield:
 - **Add to Source.** This places the value in the Source list box of the Action Properties pane.
 - **Copy to Basis.** This is used for ITERATE or IF/THEN/ELSE.
 - **Add to Destination.** This places the value in the Destination list box of the Action Properties pane.
- 5 Close the Output Message Format panel. If the Output Message Format panel is closed, then the next time it is opened, it is restored to its last location.

Temporary variables

Right-click in the Source or Destination panes to open the Temporary Variable List. From this list, select from a list of existing temporary fields for source and destination values.

You can also create your own temporary field by giving the field a name that begins with "@."

"@" is also used to create new temporary values when specified in a destination value list.

Constants

You can create a constant by preceding a value with an equal sign.

A tilde (~) is only valid on source values. This indicates that the source value was selected from the output record.

Default values

A default value is used if an input value is not available. A default value can be a hardcoded constant or a special predefined value.

For example:

- @date: Mon May 23 13:01:02 2016
- @day: Mon

- @month: Mar
- @mday: 23
- @time: 13:01:02
- @y4: 2016
- @y2: 16
- @now: Current date and time as a date type
- @null: Fills output field with pad characters

Setting active null

In HL7, a message field can contain an active null value, which is represented by quotations in the field (""). The interpretation of this value is that the receiver should clear the existing value in that field. This is in contrast with an empty field, so that the field in the receiver's side should remain as-is. To set the message field as active null, that is, the string "", during translation:

- 1 In the Source panel for the COPY action, right-click and select **@active_null** from the **Temporary Variables** menu, and then select the **Destination** field.
- 2 After translation, the **Destination** field is shown as "" for @active_null.
- 3 Clear the @active_null value by copying "@null" or "=" to the **Destination** field. This causes the field to become empty.

"%" variable names

Developers can use any name for "%" variables. For example, %nte1.

This name pattern is described by `regexp`.

From:

```
%([gsflu][0-9]+)
```

To:

```
%([a-zA-Z])([0-9])
```

The maximum variable name length is 16.

Configuring a database schema format

When a database schema is a format, you must also select a connection and table schema.

The **Connection** menu lists all configured visible database connections under the current site and master (primary) site. The master site connection is shown in *italic*.

A "visible database connection" is one that has visible table schema. "Visible table schema" means that the table schema has visible columns and is saved as a `.vrl` file.

- 1 On the **Choose File Formats** dialog box, select a connection from the **Connection** menu. Multiple selections are permitted.
 - 2 Select a table schema by clicking the **Table Schema** button. In the **Select Table Schema** dialog box, click **Add** to open a list of available table schemas.
 - 3 From the **Table Schema List** dialog box, make a selection and click **Apply** to place the selected table schemas on the **Select Table Schema** dialog box.
 - 4 Click **OK** on the **Select Table Schema** dialog box to populate the **Table Schema** field of the **Choose File Formats** dialog box.
- The database schema format tree's functionalities are the same as the `.vrl` format.

XML support and characteristics

Native support is provided for XML as a message format. This support follows the recommendations of the W3C ("World Wide Web Consortium").

XML handling is located at three points in the Translation Configurator:

- Format selection. This is in the **Choose File Formats** dialog box.
- Format presentation: Input Message Format and Output Message Format panels.
- Path string generation: Input Message Format and Output Message Format panels.

XML users must be familiar with their actual runtime message structures, as with the other formats.

XML messages fall in to the HMD, or Hierarchical Message Data, category of message formats. The XML DOM, or Document Object Model, considers the elements of XML documents to fundamentally exist in parent/child tree relationships.

The full W3C recommendation for XML includes new organizational concepts, including:

- Conditionality: An XML message element may exist in a relationship with another contender element for the same location in the message.
- Infinite recursive constructs: A message element can refer to itself.

XML also recognizes a fixed sequential organization in which one message element must follow another message element within the message. It also generalizes the concepts of optionality and repeatability under the heading of cardinality.

GUI behavior

The Translation Configurator GUIs share certain characteristics, including:

- Resizing panes
The Action Properties pane and Input/Output tree view can be vertically resized by dragging the mouse between the panes.
See [Resizing panes](#) on page 776.

- **Search functions**
Search criteria are case insensitive and substring match. Search tools are located on the toolbar.
See [Search functions](#) on page 776.
- **Copy/Paste function**
Using the copy/paste function, you can copy and paste objects between documents in the same or different IDE instances running on the same machine.
See [Copy/Paste function](#) on page 777.
- **Reconfigure feature**
If you open an XLT file where the input or output format file does not exist, a message box opens.
See [Reconfigure feature](#) on page 777.

Resizing panes

The Action Properties pane and Input/Output tree view can be vertically resized by dragging the mouse between the panes.

When the height of the Action Properties pane is less than its minimum height, you can scroll using the vertical scroll bar.

Source and Destination are resized proportionally based on the original ratio with respect to the Action Properties pane size.

When the Action Properties pane is horizontally resized, Source and Destination are horizontally resized. When it is vertically resized, Source and Destination are vertically resized.

Search functions

Search criteria are case insensitive and substring match. Search tools are located on the toolbar.

A search can be confined to the action table, or input/output format tree views:

- **Action:** The search is confined to the action table.
- **Inbound:** The search is confined to the input format tree view.
- **Outbound:** The search is confined to the output format tree view.

When selecting a search scope, specify some characters into the text box and click **Search**. The IDE searches the string in the selected scope and highlights the node or action that first matches the string.

Only the values in the Operation and Source/Destination columns are compared when actions are searched. After a string is found that matches the search string, the entire row of the action is highlighted.

When searching the trees, the tree node is expanded if the matching node is hidden. For the XML format, the IDE searches for the matched string only within the loaded tree views.

If nothing is found, then a message box opens saying, for example: Inbound message item not found.

Copy/Paste function

Using the copy/paste function, you can copy and paste objects between documents in the same or other IDE instances running on the same machine.

This function is not limited to the current configuration file, but can cross several configuration files, so that you can copy across other documents:

- Within the same sites.
- Across several sites within the same IDE instance.
- Across several IDE instances.

After saving the `xlt` file, you can verify it with **File > Compile**.

In addition to doing a copy/paste in the same file, you can copy/paste the operation entries in the same site across several files. The Translation Configurator verifies whether the pasted action item's Input/Output Message Format are the same as the message formats of the pasted file.

Affected paste actions include COPY, CONCAT, TABLE, MATH, PATHCOPY, and CALL.

If you copy/paste the action entries between two sites, then the IDE prompts with a confirmation dialog box.

Reconfigure feature

If you open an XLT file where the input or output format file does not exist, then a message box opens. This message box states that it cannot find the file or that the file does not exist. At this point, you can reconfigure the XLT file or to cancel the open operation.

Error handling

For the COPY, CONCAT, PATHCOPY, TABLE, MATH, BULKCOPY, CALL, and IF actions, you can specify how errors are handled:

- Skip: Ignore the command.
- Pad: Pad the empty spaces in a fixed-length field.
- Error: Send the error to the error database.

Translation scripting

Translation scripting includes:

- [Pre and post procedures](#) on page 778
- [Selecting Tcl code](#) on page 779
- [Calling Java/Python/Javascript class from an xlate](#) on page 779

Pre and post procedures

Pre and post procedures are on different tabs, but use the same GUI, in the Action Properties panel.

In the **Proc** text field, the Tcl and Java procedures use the same text field.

- Select **Tcl** to specify the Tcl procedures into the text field and click the **Edit** button to open the Tcl Script Editor.

Right-clicking in the field and clicking **Insert** opens the **XLTP Properties** dialog box.

- Select **Java** to specify Java procedures. Click the **Edit** button to open the **Java Pre/Post Procedure Settings** dialog box.

Right-clicking in the field and clicking **Edit** opens the **Java Pre/Post Procedure Settings** dialog box.

Pre proc example

This example shows how to write a pre-procedure in a Translation Configurator COPY action. For example, pulling something from a source field, doing an operation on it, and storing it to the destination field.

A post procedure is similar.

```
#####
# Name:      xltPDTM
# Purpose:   Convert HL7 DTTM to discrete fields for XML
# UPoC type: xltP
# Args:      none
# Notes:
#           This proc is to be used with the xmlNullConvert proc.
#           The null convert proc adds the hex 166 in place of
#           double quotes for active nulls.
#           All data is presented through special variables. The initial
#           upvar in this proc provides access to the required variables.
#
#           This proc style only works when called from a code fragment
#           within an XLT.
#
proc xltPDTM {} {
    upvar xlateId      xlateId      \
        xlateInList   xlateInList   \
        xlateInTypes  xlateInTypes  \
        xlateInVals    xlateInVals   \
        xlateOutList   xlateOutList  \
        xlateOutTypes  xlateOutTypes \
        xlateOutVals   xlateOutVals
    set newList ""
    set aN [ctype char 166]
    if {[string equal [lindex $xlateInVals 0] {}]} {
        set iv $xlateInVals
        if {[string equal [lindex $xlateInVals 0] [ctype char 166]]} {
            set iv "$aN$aN$aN$aN$aN$aN$aN$aN$aN$aN$aN$aN$aN$aN$aN$aN"
        }
        set yrs [csubstr [lindex $iv 0] 0 4]
        set mon [csubstr [lindex $iv 0] 4 2]
        set day [csubstr [lindex $iv 0] 6 2]
        set hrs [csubstr [lindex $iv 0] 8 2]
        set min [csubstr [lindex $iv 0] 10 2]
        set sec [csubstr [lindex $iv 0] 12 2]
        if {[string equal $hrs $aN$aN] && $hrs > 24} {
            set hrs 00
        }
        if {[string equal $min $aN$aN] && $min > 59} {
            set min 00
        }
        if {[string equal $sec $aN$aN] && $sec > 59} {
            set sec 00
        }
    }
}
```



```

    }
    set outputlist [list $yrs $mon $day $hrs $min $sec]
    foreach element $outputlist {
        if {[string equal $element {}] && ![string equal [lindex $xlateInVals 0] [ctype char
166]] } {
            } else {
                lappend newlist $element
            }
        }
        set xlateOutVals $newlist
    }
}

```

Selecting Tcl code

- 1 On the **Pre Proc** or **Post Proc** tab, click **Edit** to open the **Tcl Script Editor**, where you can edit existing files or procs.
- 2 In the **Tcl Script Editor**, right-click in the panel and select **Toggle Insert text** to open the **Tcl Code Fragment Selection** dialog box.
This is where you select Tcl code for your script.
- 3 Select a Category. This populates Function with the available options.
- 4 Double-click a Function to add your script.
Usage and a description display in the field at the bottom of the dialog box.
- 5 Click **Apply** on the **Tcl Script Editor**.
This inserts the script into the Proc field of the Translation Configurator.

Calling Java/Python/Javascript class from an xlate

When running code outside of the engine, the programmer is responsible for memory and file management, process blocking, error handling, and so on.

The Java compiler JDK (`javac`) is not provided by the system.

Classes should be compiled with the same version, or earlier version, used by the system.

The classes can be compiled on another computer and then transferred to the system server. You can also install Java JDK on the system server and compile the class with the `javac` command.

After a Java class is compiled, place the class into the site `java_uccs` directory.

Run java classes with `exec` from Tcl procs and xlate Tcl fragments using the command `exec java class name`. Do not include the `.class` file extension with the class name. The Tcl code captures and uses what is returned by the Java class.

Some java classes must reference Java code included in java jar files such as Java database drivers:

- The `.jar` files can be transferred to the system server.
- The CLASSPATH environmental variable must be updated to include the path to this new jar file.

You can test the Java class and a referenced java .jar file from the command line. To do this, update CLASSPATH with this command, or similar, depending on the shell:

```
export CLASSPATH=$CLASSPATH:<path to the .jar file>
```

To run a Java class and a referenced java .jar file from Tcl or an xlate Tcl fragment, use this code as an example:

```
global env
set jar [file join $::HciSiteDir java_uccs classes12.jar]
set class helloworld
set env(CLASSPATH) "$env(CLASSPATH)$jar"
set xlateOutVals [list [exec java $class]]
```

This code demonstrates how to access and modify the CLASSPATH environment variable. It also shows how to `exec` the Java class and how to capture the return data from the Java class.

Modify the code to point to your class and .jar. This code works both in an xlate Tcl fragment and in an xlate XLTP Tcl procedure.

Using the \$xlateConfigFileName global variable

For better logging, the XLTP proc has access to the current xlate file name.

Use the predefined XLTP global variable \$xlateConfigFileName in the **PreProc/Post Proc** tabs in the xlate action.

For example:

```
set xlateOutVals [list $xlateConfigFileName]
```

You can also use the predefined XLTP global variable \$xlateConfigFileName in the XLTP template:

```
#####
# Name:      __MODULE_NAME__
# Purpose:    <description>
# UPoC type:  xlt
# Args:      none
# Notes:      All data is presented through special variables. The initial
#              upvar in this proc provides access to the required variables.
#
#              This proc style only works when called from a code fragment
#              within an XLT.
#
proc __MODULE_NAME__ {} {
    upvar xlateId          xlateId
    xlateInList            xlateInList
    xlateInTypes           xlateInTypes
    xlateInVals            xlateInVals
    xlateOutList           xlateOutList
    xlateOutTypes          xlateOutTypes
    xlateOutVals           xlateOutVals
    xlateConstType         xlateConstType
    xlateConfigFileName    xlateConfigFileName
}
```

Example

```
hcixlttest -i -e ASCII -d 1 -f nl actionPreProc.xlt C:/infor_dev/cis6.1.2.0/integrator/t-  
xpm.copy/test/actionPreProc.dat
```

The output would be:

```
xlateFile is -----actionPreProc.xlt -----  
MESSAGE 1  
0(0).MSH(0) : >|^~\&|||||ORU||P|2.1<  
0(0).MSA(0) : ><  
1(0).0(0).PID(0) : >||PATIENT|||19620416|M<
```

Xlate Debugger

The Xlate Debugger, integrated into the Translation Configurator, is for users who require detailed information when developing and troubleshooting translations. You can observe the run-time behavior of the translation process and locate logic errors and addresses.

Note: When using the Xlate Debugger, the IDE does not open/close any panes. All panes remain in the same state (open/closed) when the debugger is opened/closed.

The `hcixlttest` testing command provides debugging on every action during the translation.

`hcixlttest` tests a translation configuration against an actual data message. For every input message, the Translation Configurator parses the input message according to the translation's input record specification. It then produces an output message according to the output record specification. After this, it runs the transformation actions defined in the `.xlt` file, and shows the results according to the chosen detail level.

Translation files are stored in `$HCISITEDIR/xlate` (UNIX) or `%HCISITEDIR%\xlate` (Windows).

Example

A user is developing an interface for a new downstream system. The downstream is not receiving the data in a specific field of an HL7 v2 message.

The translation has several IF and ITERATE conditions.

To debug, the user sets breakpoints in key areas in the translation to step through their logic and correct the issue.

Debug mode

Debug mode is designed for debugging the `.xlt` files. It includes an editing area and debug panel.

After the debugger is launched, Translation Configurator changes to debug mode. This automatically closes the Launch Bar and Site Manager to free the space for the debug panel.

Debug

When using TCP/IP between the client and debug server, use ports starting at 25534.

Xlate Debugger server

After launching the Xlate Debugger for an `xlt` file, the Xlate Debugger server creates a PID file (`pid_xltFileName`) and a port file (`cmd_port_xltFileName`). These are created according to the `xlate` file under `$HCISITEDIR/Xlate(UNIX)` or `%HCISITEDIR%\Xlate (Windows)`.

When the Debugger exits, the temp files are removed.

The GUI checks `cmd_port_xltFileName` before launching the Xlate Debugger. Only one debugger can be launched for a single `xlt` file.

Modes

Available modes are Configuration and Debug. You can switch between modes by clicking **Switch Mode**.

- Switching from Configuration mode to Debug mode hides the Launch bar and Site Manager and displays the debug-related components.
- Switching from Debug mode to Configuration mode restores the Launch Bar and Site Manager and hides the debug-related components.

Server Administration debug connection

The XLT Debug Traffic pane of the **Host Server Advanced** tab contains the **Host Server routes traffic** option. This is used to configure the XLT debug connection. When this is cleared, it is a direct connection. This option is disabled by default.

If there is a firewall between the client and server, then you must select **Host Server routes traffic**. Otherwise, the GUI attempts to open a port.

Action breakpoint column

An action breakpoint suspends the execution of the `xlate` action at the location where the breakpoint is set. Breakpoints can be disabled as required.

Breakpoints are indicated by action indexes that start from 1. Each action, including disabled actions, has an index.

- If any action is removed, then all indexes of the actions behind the removed action are decreased by 1.
- If any action is inserted, then all indexes of the actions behind the inserted action are increased by 1.
- The Index column can be resized as required.

The action breakpoint column shows the breakpoints and the Index column shows the index of the `xlate` action.

Breakpoints cannot be placed on a COMMENT or ELSE action. COMMENT does not behave practically at runtime and is skipped, and ELSE is not a real action.

A disabled action can be set for a breakpoint, when the Xlate Debugger is not running, since they can be enabled later. The Xlate Debugger does not pause on disabled actions.

A breakpoint cannot be set for a disabled action when the Xlate Debugger is running.

The Xlate Debugger highlights the next row to be run in green. If this row is also selected, then the color changes to a darker green.

You can add a breakpoint by double-clicking the breakpoint column. This also removes an existing breakpoint.

You can also add or remove a breakpoint using the context menu on the action table or the **Debug** menu:

- Select **Set Breakpoint** to set a breakpoint to the selected action.
- Select **Remove Breakpoint** to remove the breakpoint.

These are unavailable until an action is selected.

For multiple selections:

- If all of the selected actions have breakpoints, then only **Remove Breakpoint** is enabled.
- If all of the selected actions do not have any breakpoints, then only **Set Breakpoint** is enabled.
- If some of the selected actions have breakpoints, then **Set Breakpoint** and **Remove Breakpoint** are enabled.

Breakpoints remain with the actions. For example, when an action is shifted due to other actions being added or removed, the breakpoint moves with the action.

Disable a breakpoint by selecting **Disable Breakpoint** on the context menu.

Enable a disabled breakpoint by selecting **Enable Breakpoint** on the context menu.

For the multiple selections:

- If all of the selected actions have enabled breakpoints, then only **Disable Breakpoint** is enabled.
- If all of the selected actions have disabled breakpoints, then only **Enable Breakpoint** is enabled.
- If some of the selected actions have enabled breakpoints and some have disabled breakpoints, then **Enable Breakpoint** and **Disable Breakpoint** are enabled.

Debug panel

The Debug panel displays debugging information, for example, input values, user-defined temporary variables, iterate variable values, and so on. This contains these views that are shown as tabs:

- Breakpoints
- Variables
- Expressions
- Input Values
- Console
- Source Message
- Target Message

For each pause on the xlate action, these visible views are updated with the latest data.

You can show/hide views by selecting from the **View** menu.

The **View** menu options are enabled only when the current Translation Configurator is in Debug mode. If the current GUI is in Configuration mode, then these options are unavailable.

You can also can hide those views by clicking the close button on the view tabs.

Breakpoints view

This lists all breakpoints in the current debugging session.

Remove the marked breakpoints by clicking **Remove the marked breakpoints**.

Enable all breakpoints by clicking **Enable all breakpoints**.

Variables view

This shows the values for the current user-defined temporary variables and the `ITERATE` variables in a table:

- The Name column lists the variable names.
- The Value column shows the corresponding values.

Right-click in a **Name** or **Value** field to open a menu of "@ vars."

Note: The value of system temporary variables and `ITERATE` variables cannot be modified. The value cell is not editable for these two types. The context menu for system temporary variables is available in the Variables view.

Expressions view

This is used for inspecting the values of specified address paths, temporary variables, or `ITERATE` variables.

Clicking **Add watch expression** creates a new empty row. Double-clicking in a cell opens it for editing, where you can specify an address path, temporary variable name, or `ITERATE` variable name into the cell.

When you press **Enter**, the GUI send a request to the debugger for the value of the nonempty expression. The response is displayed in Value.

Add a tilde (~) before the path address to inspect the value of the address path of output message.

The Value cell is the same as the Variables view, and is editable except for system temporary variables and `ITERATE` variables.

To simplify specifying an address path, add the corresponding path of the tree node as a watch expression. You can do this by dragging an address and dropping it into the Expression table.

You must first add a new row by clicking **Add watch expression**, then you can drag an address into the cell.

A tree node can be dropped into any editable text box.

When debugging, if a tree node is dropped into a table, the GUI sends a request directly to the debugger and the value is shown. There is no requirement to press **Enter**.

If you are dropping an address node into a cell in editable mode, then you must press **Enter** to get the value.

Input Values view

This displays the input values for the currently paused xlate action.

Double-clicking a Value cell makes it editable, where you can modify the value. The modification becomes effective when you press **Enter**.

The Name cell is not editable.

Console view

This displays what is printed to the standard output and standard error by the command line `hcixlttest`.

When an error is encountered, detailed information is shown in Console view. In this situation, stay in the Xlate Debugger after the debugging session has finished and check the Console view for information.

To do a search, input a keyword into the text field on the toolbar and click **Search**. The item that matches the search string is highlighted.

Click **Clear the console** to clear the panel.

Source Message view

This displays the encoded input message at the specified **Detail level**. Although there may be multiple input messages in the translation, this view only displays the currently running message.

You can change the configured detail level by switching the selection in the **Detail Level** combo box. After the detail level is changed, the displayed source message is changed to the corresponding detail level.

In the Xlate Debugger, the detail level in the Source/Target Message view is the same as the setting in the **Xlate Debug Configuration** dialog box. By default, the detail level is "1" in the **Xlate Debug Configuration** dialog box.

To do a search, input a keyword into the text field on the toolbar and click **Search**. The item that matches the search string is highlighted.

The text field used to show the source message is not editable.

Search criteria are case insensitive and substring match.

Target Message view

This displays the partially completed target message at the specified **Detail level**.

To do a search, input a keyword into the text field on the toolbar and click **Search**. The item that matches the search string is highlighted.

The text field used to display the partial target message is not editable.

The target message can be automatically or manually refreshed. Click **Refresh target message** to manually refresh.

This is changed in **Client Preferences > Xlate Config** by selecting or clearing **Automatically refresh the target message during xlate debugging**.

This option is also located in `client.ini`:

```
xlt_debug_auto_refresh_target_msg=true/false
```

Debug information is saved on the client side as:

```
user.home/.clide/xlate/hostname_version_siteName_xltFile.debug
```

This file is saved as the sectioned `ini`:

```
[configuration]
msg_file_name=
msg_file_style=
msg_file_encoding=
source_msg_detail_level=0/1/2/3
target_msg_detail_level=0/1/2/3
record_num=1/ALL
show_field_name=true/false
detect_leak=true/false
separator_opt=
save_file_name=
save_file_style=
pretps=
posttps=
[breakpoint]
index=1 2 3 4 5 6
status=1 0 1 1 1 1
[expression]
expression=address1 address2 temporary1 temporary2
```

Debug buttons

Clicking **Debug** launches the debugger with the most recently configured debug configuration. If this is the first time for the xlate file to be debugged, this button is unavailable since there is no available debug configuration. This is enabled if the xlt file has been previously debugged. When an xlate file is reopened, **Debug** is disabled.

Clicking **Debug with Configuration** opens the **Xlate Debug Configuration** dialog box.

This table shows the available parameters:

Parameter	Description
Data File	Click the folder button to open a file browser. Select the data file from the \$HCIR00T folder.
Format	Click the arrow to open a menu of formats in which to read the input Data File .
Encoding	Select different multi-byte encodings from the menu.
Detail Level	Click the arrow to open a menu of detail levels. Detail levels go from 0, for raw, unparsed data, to 4, for most detail.

Parameter	Description
Process One Record	Reads the selected data file and processes only the first message in the file.
Process All Records	Reads the selected data file and processes all the messages in it.
Pre Xlate TPS	<p>Click Edit to open the TPS Editor. Use this dialog box to select the TPS procedures to run before message translation.</p> <p>Click Add to select the procedures or Java class. Then, specify any arguments.</p> <p>Click OK when finished.</p>
Post Xlate TPS	<p>Click Edit to open the TPS Editor. Use this dialog box to select the TPS procedures to run after message translation.</p> <p>Click Add to select the procedures or Java class. Then, specify any arguments.</p> <p>Click OK when finished.</p>
Save To	Saves the output message to a file. Click the folder button to open a file browser. Select the file in which to save the output message or specify the file name in the field. This file is saved in the \$HCISITEDIR folder.
Line Termination Format	<p>Click the arrow to open a menu of formats in which to save the output message:</p> <ul style="list-style-type: none"> • Newline Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser. • Length-Encoded reads the first 10 characters to determine the length of the first message. Then it reads that many characters into a message, and sends it to the parser. • EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.
Show Field Names	Select the check box to show field names.
Leak Detection	Select the check box to check for memory leaks.

Parameter	Description
Advanced	Opens the Advanced Options dialog box. Separator Options parse the message with the specified separator characters (Inbound) and encodes the message with specified separators (Outbound). If a separator is not specified, the default is used.
OK	Clicking this on the Xlate Debug Configuration dialog box launches the current debug configuration. Simultaneously, the Translation Configurator switches to Debug mode. If the debugger cannot be launched, an error message opens to indicate the cause.

After the debugger is launched, it pauses at the first `xlt` action if there is no breakpoint.

When the debugging session begins, clicking **Debug** opens a Confirm message. Click **Yes** to restart the Xlate Debugger with the current configuration, or **No** to cancel the operation.

To maintain synchronization of the `xlt` file between the GUI and the Xlate Debugger, you must save modifications before launching the Xlate Debugger.

Deactivation and activation

To prevent an action from running in the message translation, you can comment it in the Translation Configurator using several methods:

- Using the **Edit > Deactivate** menu option.
- Selecting **Deactivate** from the right-click menu.
- Using the **Deactivate** toolbar button.

Uncomment it by selecting the **Activate** option, so that it can be performed in the message translation.

Commented lines are rendered in gray.

Note: In the Site Document, disabled actions are displayed in gray.

Comment criteria

For those actions that can have sub-actions, such as IF, ELSE, and ITERATE, if they are commented out, all sub-actions are commented out. They cannot be uncommented individually unless the ancestor is uncommented.

When the ancestor is uncommented, all the sub-actions return to their original status.

For example:

- If there is a commented COPY under an uncommented IF, and you comment out IF, then COPY is commented. If you uncomment IF, then COPY keeps commented.
- If there is an uncommented COPY under an uncommented IF, and you comment out IF, then COPY is commented. If you uncomment IF, then COPY becomes uncommented.

- If you comment out an IF that has an ELSE, then ELSE is commented out. It cannot be uncommented individually unless the corresponding IF is uncommented. You can still individually comment an ELSE when IF is uncommented.

When you copy or cut an action and paste it, the pasted action keeps its original status. The pasted action stays commented when it is pasted under a commented ancestor.

For example:

- When you copy or cut a commented action that is on the top level, no matter where you paste it, the pasted action stays commented.
- When you copy or cut an uncommented action and paste it into a commented IF, ELSE, or ITERATE, the pasted action becomes commented.
- You can copy or cut an action under a commented IF, ELSE, or ITERATE. After this, you can paste it into the top level or under an uncommented IF, ELSE, or ITERATE. The pasted action keeps the same state it had before its original ancestor was commented. If you paste it under a commented IF, ELSE, or ITERATE, then the pasted action stays commented.

Move operations such as **Move Up** or **Move Down** can be selected on an action. The state of the action follows the same rule as those after the copy/paste operation is applied.

If no row is selected in the action table, then both **Activate** and **Deactivate** are not available.

For a single selection, both **Activate** and **Deactivate** are not available in these instances:

- The action has a commented ancestor.
- The action is a commented ELSE and its sibling IF is also commented.

An exception to this is when **Activate** is enabled on a commented action and **Deactivate** is enabled on an uncommented action.

For multiple selections:

- If any of these selections belong to one of the single selection cases, then both **Activate** and **Deactivate** are not available.
- If all the selections are commented, then **Activate** is enabled and you can uncomment those selections by clicking it.
- If all the selections are uncommented, then **Deactivate** is enabled and you can comment those selections by clicking it.
- If all the selections are not commented, then **Deactivate** and **Activate** are enabled.

Note: You can still change the properties of a commented action, similar to editing an uncommented action. The change is saved into the `.xlt` file, similar to saving the change of an uncommented action.

When you select the **File > Compile** menu option to compile a translation file, it calls the `hcixlttest xltfile` command, which compiles the `.xlt` file.

If a translation file has an invalid action option, address, and so on, then an error is reported. If an invalid action is commented out, or it has a commented ancestor, then the error is not reported by the `hcixlttest` command. That is, it is not validated.

Actions

Translation actions are tools that the Translation Configurator applies during the translation process. They convert the contents of the source message from its original format to the destination message format.

Selecting an action tool creates a new empty action which automatically loads into the Action Properties pane for editing.

Right-clicking an action opens a menu of options.

Creating an action

After selecting the file formats, create a new action.

- 1 Select the New Action from the toolbar or from the right-click menu in the layout panel. The mode determines where the action is placed.
 - **Append** places the action at the end of the currently selected branch, as the last child of the currently selected action's parent..
 - **Before** places the action before the current selection on the Translation Configurator. The action is inserted relative to the currently selected action's parent.
 - **After** places the action after the current selection on the Translation Configurator. The action is inserted relative to the currently selected action's parent.
- 2 Select the action by clicking the appropriate tool on the New Action toolbar. This places a new empty action at the location. Action choices are:
 - **COPY** copies the source values directly to the destination fields.
 - **CONCAT** concatenates two or more input fields into a single output field.
 - **TABLE** translates the source values according to the specified translation lookup table.
 - **ITERATE** defines an iterative block of translation actions. The block repeats every time the specified basis is encountered in the input record. The basis Type can be a record variant Field, Group, List, or Segment.

When Type is user, the Basis can be a user-defined variable that stores a list.
 - **MATH** performs the specified math operation on the source values. Specify at least two source values for a math operation. One or more can be a constant value.
 - **ADD** adds the source values.
 - **AVG** takes the average of the source values.
 - **DIV** divides the source values.
 - **MAX** takes the maximum of the source values.
 - **MIN** takes the minimum of the source values.
 - **MOD** takes the remainder after dividing the source values.
 - **MUL** multiplies the source values.
 - **SUB** subtracts the source values.
 - **STRING** provides string operations, such as Trim, Substring, Concat, and others.
 - **COMMENT** includes a text comment with the translation definition.
 - **BULKCOPY** copies every common input field to output.
 - **PATHCOPY** copies, clears, and replicates information that is grouped at almost any level of association.

- **CALL** calls a Tcl procedure to perform a set of operations on the source values.
 - **CONTINUE** generates a message from all the translated elements up to the Continue, and places it on the disposition list as CONTINUE. This includes further processing of the generated message.
 - **DATECOPYOPT** defines how to handle date copies.
IF provides controls for defining a condition and the action to be taken if that condition exists.
 - **SEND** generates a message from all the translated elements up to the send, and transmits that message to the outbound connection. This action permits the generation of multiple outbound messages from a single inbound message. The SEND and CONTINUE order is important. This bypasses normal processing and sends the generated message to the Partial Queue. Otherwise, it is the same as CONTINUE.
 - **SUPPRESS** transmits the message. This action permits the generation of multiple outbound messages from a single inbound message. It suppresses processing of the original input message beyond the translation phase. SUPPRESS is the last translation operation if SEND or CONTINUE operations generate multiple outbound messages from one inbound message.
 - **BREAK** is used to break a loop or to skip iterates.
 - **INCLUDE** can use a reference in the xlate, where you create one xlate for the PID translation and refer to it in the other xlates.
- 3 After selecting the action, edit the details of the operation by clicking the action and editing the Source/Destination values, adding procs, and so on.

COPY

Copies the source values directly to the destination fields, providing direct value copy from a list of inputs to a list of outputs.

Value list entries are matched on a subfield by subfield basis.

Note: A COPY always copies something to the output, so that if nothing is specified, the input value is copied.

Fetching value

Note: In this section, "field" and "subfield" are used. This is for HRL/VRL/FRL formats. For HMD formats, replace "field" with "component" and "subfield" with "subcomponent."

The fetching value based on a field-level address has changed from CIS5.8 to CIS6.0 and later versions.

In pre-CIS6.0 versions, when setting a field-level address as the source in an action and the field had subfields, the engine would expand the field. After this, the engine retrieves all the subfield's content and put the values in a list.

This led to issues:

- If there was only one destination, then only the first subfield content was copied to that destination.
- If there are several destinations, then the subfield's content was copied to incorrect destinations.

A change has been made in how the engine retrieves values from a field-level address as the source side in a translation action.

When the subfield-level address is used as a source of a COPY action, the engine retrieves the whole subfield content. This includes all sub-subfields, as a string. It does not copy only the first sub-subfield.

Then, the whole content as a string is assigned to `xlateInVals`.

When the field-level address is set as a source, the engine retrieves the whole field content. This includes all its subfields and the subfield separator characters as a string, so that the whole field content is copied to the destination.

JSON RAW Copy

When doing a JSON RAW Copy action, if the source data style is “Scientific notation” and the destination node type is “number” and the result has a decimal, then the result displays six decimal places. Otherwise, there is no decimal.

Example 1:

If the source node type is a number and the value is “0.8e-3”, then after a COPY xlate action the result is “0.000800”.

Example 2:

If the number is “0.8e+2”, then after a COPY xlate action the result is “80”.

COPY example: subfields

VRL has a field `Test` which has three subfields. The subfield separator char is “-”.

Field `Test2` does not have any subfields.

The input content is:

- `Test: >A-B-C<`
- `Test2: >D<`

COPY example: One source and one destination

```
{ { OP COPY }
  { ERR 0 }
  { IN Test }
  { OUT @tmp }
}
```

Before CIS 6.0:

- The `@tmp` in CIS5.8 was `>A<`.
- Only the first subfield was copied.

After CIS 6.0:

- The `@tmp` in CIS6.0 is `>A-B-C<`.
- The entire field content is copied.

COPY example: Two sources and two destinations

```
{ { OP COPY }
  { ERR 0 }
  { IN {Test Test2} }
  { OUT {@tmp @tmp2} }
}
```

Before CIS 6.0:

- The @tmp was: >A<.
Only the first subfield was copied.
- The @tmp2 was: >B<.
The second subfield was copied to this destination, which is expected to be the content of field Test2(D).

After CIS 6.0:

- The @tmp is: >A-B-C<.
The whole field content is copied.
- The @tmp2 is: >D<.
The content of field Test2 content is copied.

The content of xlateInVals in Pre Proc has also been updated:

- Before CIS 6.0, xlateInVals was a list.
- After CIS 6.0, xlateInVals is a string.

```
{ { OP COPY }
  { ERR 0 }
  { PRE {
    #Note: Shows $xlateInVals
    puts "xlateInVals is: $xlateInVals"
    lassign $xlateInVals a b c
    puts "$a:$b:$c"
  }}
  { IN Test }
  { OUT @tmp_test }
}
```

Before CIS 6.0:

- xlateInVals was list A B C.
- Using lassign assigned the subfield's content to variables a,b,c.

This displayed:

- xlateInVals is: A B C
- A:B:C

After CIS 6.0:

- xlateInVals is a string A-B-C.
- Using lassign assigns A-B-C only to variable a, and makes b,c empty strings.

This displays:

- `xlateInVals` is: A-B-C
- A-B-C::

This change happens on HRL/VRL/FRL and all HMD formats.

COPY example: Updating old translation file

This example shows how to update the old translation file to make it work under the new retrieve behavior.

To copy only the subfield content, use the subfield-level address as the source instead of the field-level address.

For example:

```
{ { OP COPY }
  { ERR 0 }
  { PRE {
    #Note: Shows $xlateInVals
    puts "xlateInVals is: $xlateInVals"
  }}
  { IN Test.[1] }
  { OUT @tmp_test }
}
```

This copies only the second subfield content `B` to the destination; `xlateInVals` is `B` only.

To copy the whole field content, but have written a pre-proc to handle the `xlateInVals` list, keep using the field-level address in the source. Then, update the pre-proc.

To update your proc:

```
{ { OP COPY }
  { ERR 0 }
  { PRE {
    #Note: Shows $xlateInVals
    puts "The original xlateInVals is: $xlateInVals"
    regsub -all {-} $xlateInVals { } xlateInVals (Replace the sub-field separator char
with space, so that xlateInVals will become a list.)
    puts "The xlateInVals after regsub is: $xlateInVals"
    lassign $xlateInVals a b c
    puts "$a:$b:$c"
  }}
  { IN Test }
  { OUT @tmp_test }
}
```

This displays:

- The original `xlateInVals` is A-B-C.
- The `xlateInVals` after `regsub` is A B C.
- A:B:C

CONCAT

This action concatenates two or more input fields into a single output field.

This action is useful when you have more subfields in an input field than in the target output field. The remaining input subfields are concatenated into the last output subfield.

You can also concatenate constants and subfields into a destination value.

In the **Separator** field, specify the character to place between the fields which are being concatenated.

For example, three fields must be separated with a comma:

- PID.lastname
- PID.firstname
- PID.middleinitial

To do this, select CONCAT and specify a comma (,) in the **Separator** field. This results in lastname, firstname, middleinitial.

TABLE

This action translates the source values according to the specified translation lookup table. This action accepts a single input value and produces a single output value.

Use the Lookup Table Configurator to configure the lookup tables used to convert message data. Lookup tables perform code translations within one or more translation configurations.

If the table is bi-directional, then you can specify which side of the table is used in looking up the input values.

A Lookup Table is called up within a translation configuration and used in the Table action for a specific field.

When using a Lookup Table, you must know the name of the table to use. You must also know which side of the bi-directional table to use for the input values.

For **Table**, click the arrow to select the table file from the menu.

For **Side**, click the arrow to select Input or Output.

Note: All tables must have the .tbl extension to be viewed in the Site Manager and lists within Translation Configurator. If these tables are created from within the GUI, then they already have the .tbl extension. Sites that are brought forward from previous versions that do not use the .tbl extension must have the table files renamed with the .tbl extension. The references must also be updated in the translation that uses the table.

Open Lookup Table

When editing a TABLE action in an xlate, you can select **Open Lookup Table** to open the lookup table.

This is also on the right-click menu in the action panel and **Edit** menu.

This option is disabled when the **Table** is empty.

Database table

The database lookup table action is supported in the existing TABLE action. A database table action indicates it is a TABLE action where the selected table is a database table.

The Lookup Table Configurator is responsible for configuring a new lookup configuration into a `.tbl` file. The Translation Configurator applies this `.tbl` file in a TABLE action.

A `.tbl` file is used to record the database table configuration.

The type that is defined in `.tbl` is used to distinguish the traditional Cloverleaf table and Database Lookup table.

The database lookup configuration tool has same control options as the table lookup configuration tool.

Translation Configurator usage

The **Database Lookup** option in the Lookup Table Configurator uses the TABLE action to do the lookup:

- The table uses column name and value address mapping.
- Multiple sources to multiple destinations is supported.

When the selected table configuration is a database and it has multiple In/Out columns, the corresponding source/destination list display as a table component.

The table has two columns:

- Input name/Output name
These list all the column names of the In/Out fields of the `.tbl` configuration.
- Format field
This is used to fill the correspondent message format field address. All format field cells must be populated; otherwise, the GUI prompts an error message.

Double-clicking an input/output message format tree node inserts the field address into the selected format field cell.

Under the runtime, according to the source settings, the translation engine uses the value of the specified input message format field. It then replaces the placeholder in the `.tbl` SQL statement. It also runs it to do the database lookup.

After the result is returned, the translation engine populates the specified output format field with the corresponding output value. This is according to the destination settings to compose the output message.

tbl file

The `$HCISITE\Tables\XXX.tbl` file is used to store the database lookup configuration.

For example:

```
# Translation database lookup
#
prologue
  who:      18269
  date:     July 1, 2013 2:45:17 PM CST
  outname:  output
  inname:   input
  bidir:    0
  type:     dbl
  version:  5.0
```

```

end_prologue
#
dflt_passthrough=0
dflt=
dflt_encoded=false
#
dbconnection=dbconnectionName
table_name=patient_table
sqlstatement=select * from patient_table where patientid =<patientid> and date=<date> and doctorid=<doctorid>
in_column_name= patientid,date,doctorid
out_column_name= patientid,date,doctorid,patientname,doctorname,result

```

dblookup Tcl command usage

The `dblookup` Tcl command can be used in engine Tcl UPoCs for the translation thread and in the inbound/outbound protocol thread UPoCs.

`dblookupdestroy` is not required when using `dblookup` at engine runtime. The engine automatically destroys the resources that were used by the Database Lookup table during shutdown.

- `dblookup ?-maxrow ?count?? ?-metacolumnname? table value ?value...?`

This looks up database data through a Database Lookup table that is based on the given value. The table is a Database Lookup file name. The file extension can be omitted.

The input values that are provided by `dblookup` are set as the IN column values one-by-one. The input value account equals the IN column account that is defined in the `.tbl` file. If they are not matched, then an error is prompted and the DB Lookup terminated.

- `-maxrow ?count?`

This sets the maximum returned row count. By default, at most one record is returned from the `dblookup` command. With this option, the specific `count` records are returned. To get all the selected records, set this option without `count`.

- `-metacolumnname`

This gets the column names from result set metadata. With this option, both the selected database data and column names of result set metadata are returned in a keyed list.

- `dblookupdestroy table`

This destroys the related resources. For example, the JNI context and database connection, used by `dblookup`. This command returns an empty string.

TABLE examples

```

hcitcl>dblookup test_simple.tbl key01
test1

hcitcl>dblookup test.tbl key01 test1
1,2014-04-01,0,test1,key01

hcitcl>dblookup -metacolumnname test.tbl key01 test1
{RSMETACOLUMNNAME {intFLD} {dateFLD} {flag} {strFLD} {ID}} {RSDATA {{1} {2014-04-01} {0} {test1} {key01}}}

hcitcl>dblookup -maxrow 2 test_rows.tbl 0
2014-04-01,0,key01,1,test1

```

```

2014-04-02,0,key02,2,test2

hcitcl>dblookup -maxrow 2 -metacolumnname test_rows.tbl 0
{RSMETACOLUMNNAME {dateFLD} {flag} {ID} {intFLD} {strFLD}} {RSDATA {{2014-04-01} {0} {key01} {1}
{test1}} {{2014-04-02} {0} {key02} {2} {test2}}}}

hcitcl>dblookup -maxrow test_rows.tbl 0
2014-04-01,0,key01,1,test1
2014-04-02,0,key02,2,test2
2014-04-03,0,key03,3,test3
2014-04-04,0,key04,4,test4
2014-04-05,0,key05,5,test5
2014-04-06,0,key06,6,test6
2014-04-07,0,key07,7,test7
2014-04-08,0,key08,8,test8
2014-04-09,0,key09,9,test9
2014-04-10,0,key10,10,test10

hcitcl>dblookupdestroy test_rows.tbl
hcitcl>

hcitcl>dblookupdestroy test
hcitcl>

```

ITERATE

This action defines an iterative block of translation actions. The block repeats every time the specified basis is encountered in the input record. This is the only way to deal safely with repeating elements.

Iteration is based on a specific element of the input record. For example, an HL7 segment, HL7 repeating group, a field name from a record, or a constant list.

When defining an iteration, these items must be defined:

- **Type** is what you are iterating over.
- **Basis** is where you are iterating.
- **Variable** is how to ensure you get all repetitions.

Iterating over a user-defined list referenced by a variable

To iterate over a user-defined list that is referenced by a variable, use the “user” iterate type (list, segment, field, group, and user are all iterate types).

For the basis, use the name of the variable. For example, @mylist.

The iterate then loops over each element in the list that is referenced by @mylist.

Example:

```

COPY =a b c d --> @mylist
ITERATE user @mylist %u1

```

In this example, %u1 contains “a” for the first iteration.

Type

Select the type of record element to use as the basis for the loop.

The basis type is one of:

- Field: Points to repeating fields on a segment

```
ITER %f1 on 0(0).PV1(0).00203
           0(0).PV1(0).00203(%f1)
```

- Group: Reflects repeating groups

```
ITER %g1 on group 1
           1(%g1).0(0).IN1(0).00426
```

- Constant List: Points to a defined list of fields

```
ITER %l1 on 0 1 2 3 4 5
           DRNAME_%l1
```

- Segment (default): Reflects repeating segments

```
ITER %s1 on 0(0).AL1
           0(0).AL1(%s1).00203
```

- User: See [ITERATE type user](#).

Variables

In the **Variable** field, specify the iteration counting variable to use. All variables must start with % and be one of:

- f (field)
- g (group)
- l (list)
- s (segment)
- u (user)

Other characters do not pass the validation.

For example, if there are three variables, use %f1, %f2, and %f3.

List ITERATE variable values

To access the List ITERATE variable value in the ITERATE action, use \$ as the prefix before the variable name.

For example, if a List ITERATE is defined with basis foo, bar, baz, and variable %l1, then \$%l1 can be used to retrieve the basis value in translation.

This is an example of a copy list value to outbound:

```
{ { OP ITERATE }
  { TYPE list }
  { BASIS {foo bar baz} }
```

```
{ VAR %l1 }
{ BODY {
{ { OP COPY }
{ ERR 0 }
{ IN {{$%l1}} }
{ OUT 1(%l1).PV1.00458 }
}
}}
}
```

Using ITERATE variables correctly

All ITERATE variables must start with % and be one of:

- f (field)
- g (group)
- l (list)
- s (segment)
- u (user).

Other characters do not pass the validation.

For example, it is incorrect using %s1 for a list. It should be %l1.

Double %s1 are used in an xlt file for another type (list and segment). This leads to double free for a variable, and causes "undefined behavior."

ITERATE type user

The ITERATE type **user** uses **u** as the variable name.

When the **Type** is user, the **Basis** can be a user-defined variable that stores a list.

A user list points to a defined list of fields. The list content can be used in an address as a repeat number. For example:

```
ITER %u1 on 0 3 5
0(0).PV1(0).00203(%u1)
```

Example: "user" iterate type

To iterate over a user-defined list that is referenced by a variable, you must use the "user" iterate type. Iterate types are list, segment, field, group, and user.

For the basis, specify the name of the variable, for example, @mylist.

The iterate then loops over each element in the list that is referenced by @mylist.

Example usage:

```
COPY =a b c d --> @mylist
ITERATE user @mylist %u1
```

In this example, %u1 contains "a" for the first iteration.

This feature was introduced in 6.2.0.

Customizing the basis in runtime

This examples uses a temp variable as the iteration user list basis to customize the basis in runtime.

- 1 Copy the values to a temp variable:

```
{ { OP COPY }
  { ERR 0 }
  { IN {0 2} }
  { OUT @basis }
}
```

- 2 Set the temp variable as the basis of an iteration user list.
- 3 Use the iteration variable as a repeat number in an address:

```
{ { OP ITERATE }
  { BASIS @basis }
  { VAR %u1 }
  { TYPE user }
  { BODY {
    { { OP COMMENT }
      { COMMENT {TODO: Insert new actions here} }
    }
    { { OP PATHCOPY }
      { ERR 0 }
      { IN 1(0).NK1(%u1) }
      { OUT 1(0).NK1(%u1) }
    }
  }
}}
```

- 4 The engine works on 1(0).NK1(0) and 1(0).NK1(2).

Auto entry

In translations, the GUI auto-populates a default variable name to the **Variable** field. For example, you must create a new action within the ITERATE, and use an address that uses the ITERATE variable. When this is completed, the GUI fills in that ITERATE variable from the basis ITERATE.

For example, an iteration is on 0(0).ORU. The ITERATE action should auto-populate the variable name to be %s1.

In another example, a field is copied within that ITERATE, 0(0).ORU.#2. Clicking on that address in the tree populates the repetition of the segment, so that it is 0(0).ORU(%s1).#2.

When using %s1 for HL7 segments, the %s1 variable is populated. Inside the ITERATE, the input/output field automatically has %s1 in the address. This also works for nested ITERATES.

For the ITERATE action, **Basis** is auto-populated when you double-click a node on the format tree. The **Variable** field value is set as %f1, %s1, or %g1 for different ITERATE types.

For actions inside an ITERATE, the Source and Destination texts address content is auto-substituted with the correct **Variable** name of the parent ITERATE action. This happens when you double-click any node within the ITERATE basis.

In addition, if you manually update the **Variable** name for an `ITERATE` action, all references to the old Variable name are updated automatically.

For some users, setting the basis and `ITERATE` variables in the `ITERATE` action can be complicated. The Auto Entry feature helps by decreasing the time to configure `ITERATE`.

Auto entry example 1

- 1 Add an `ITERATE` action, and click **PV1** (in this example, PV1 is repeated) in the Input Message Format tree. The repeated field is automatically populated as %gx (group), %sx (segment), or %fx (field). In this example, it is %s1.

x indicates the `ITERATE` variable index and is an integer starting with 1. The `ITERATE` variable is not duplicated by default. If %s1 exists, then %s2 is taken; if %s2 exists, then %s3 is taken; and so on.

- 2 Double-click the PV1 node, or select **Copy to Basis**. Type, Basis, and Variable of the `ITERATE` are automatically filled.
- 3 Select the FT1 node. This is another repeated segment.
 - a Then select **Copy to Basis**.
Variable does not change if they have same node type, for example, segment. The repeated field is %s1.
 - b Change **Basis** back to PV1.

A new **Variable** is populated if **Variable** is empty. This also happens when the node type of the format tree is not the same as the `ITERATE` action type.

- 4 Add a `COPY` action inside the `ITERATE` action and select a child node of PV1. The segment index is automatically substituted with %s1 in the address path.
- 5 Add an `ITERATE` action inside the `ITERATE` action and select a repeated child node of PV1. A sub-`ITERATE` action uses 0(0).PV1(%s1).#20 as the **Basis** and **Variable** are automatically populated.
- 6 Add a `COPY` action inside the sub-`ITERATE` action and select a child node of PV1(0).#20. The segment index and field index in the pathname are automatically substituted with the super `ITERATE` variable (%s1) and the sub-`ITERATE` variable (%f1).
- 7 Add another `ITERATE` action Basis on output node FT1. The **Variable** is %s2.
- 8 Add a `COPY` action inside the `ITERATE` action. The input side uses %s2 for pathname.
- 9 If you manually change the `ITERATE` variable index to another integer, then the references for this `ITERATE` variable are automatically updated. References can be the properties of the actions under `ITERATE`. Any substrings that entirely match the old variable are substituted with a new variable.

Updated action properties are:

- `ITERATE`: Basis
- `COPY/CONCAT/TABLE/MATH/STRING/CALL`: Pre Proc, Post Proc, Source, Destination
- `CALL`: Proc
- `IF`: Condition, ELSE
- `PATHCOPY`: Source, Destination
- `INCLUDE`: Group Prefix in Host Xlate

Auto entry example 2

This is an example of the auto entry feature, to ITERATE over the NK1 segments in an HL7 ADT message.

- 1 Click ITERATE, right-click the NK1 segment in the input window, and select **Copy to Basis**. The ITERATE properties are displayed with the correct path for the NK1.
- 2 Specify `0(0).NK1` as the basis. The ITERATE variable defaults to `%s1`.
- 3 Add a COPY statement to the ITERATE:
 - a Double-click a field in segment NK1 in the Input Message Format tree. The path is displayed in the input source view. The ITERATE variable is automatically inserted into the path, for example, `0(0).NK1(%s1).#2(0).[0]`.
 - b Double-click a field in segment NK1 in the Output Message Format tree. The path is displayed in the output source view. The ITERATE variable is automatically inserted into the path, for example, `0(0).NK1(%s1).#2(0).[0]`.

Iterating over a user-defined list referenced by a variable

To iterate over a user-defined list that is referenced by a variable, use the “user” iterate type. List, segment, field, group, and user are all iterate types.

For the `basis`, use the name of the variable. For example, `@mylist`.

The iterate then loops over each element in the list that is referenced by `@mylist`.

Example:

```
COPY =a b c d --> @mylist
ITERATE user @mylist %u1
```

In this example, `%u1` contains “a” for the first iteration.

INCLUDE

The Translation Configurator supports the INCLUDE action.

Many times, there are numerous xlates in an HL7 translation, including `ADT_A01.xlt`, `ADT_A02.xlt`, `ADT_A03.xlt`, and so on. In these xlates, the transformation for PID segment is the same. In this instance, if you have a modification on a PID field you must modify each xlate, which can be time-consuming.

With the INCLUDE action, you can use a reference in the xlate. You only create one xlate for the PID translation and refer to it in the other xlates.

There are two steps to using the INCLUDE action:

- 1 Configure the sub-transactions. You can configure these on the segment level, for example, a PID segment in HL7.
- 2 Click INCLUDE and add the defined sub-transactions in an actual transaction.

In the Translation Configurator, the INCLUDE action is accessed from the toolbar or the **Action** list.

The nested INCLUDE is supported.

Prefix substitution

You can define the prefixes in the Source and Destination fields. Prefixes are used to replace the group prefixes in sub-translation with specified ones in the current translation configuration.

The prefixes support multi-replacement pairs.

BOX Manager, Site Document, and CLAPI

The BOX Manager can package included xlt files that include other xlt files.

The Site Document supports xlt files and the INCLUDE action.

The clapi-xlate project supports the INCLUDE action as:

```
CloverXaSend extends AbstractCloverXltAction<Xainclude>
```

Example: Include a translation

Many times, there are numerous xlates in an HL7 translation, including ADT_A01.xlt, ADT_A02.xlt, ADT_A03.xlt, and so on. In these xlates, the transformation for PID segment is the same. In this instance, if you have a modification on a PID field you must modify each xlate, which can be time-consuming.

With the INCLUDE action, you can use a reference in the xlate. You only create one xlate for the PID translation and refer to it in the other xlates.

Using INCLUDE

- 1 Click the **INCLUDE** action button to add it to the Operation list.
- 2 Select the xlt file to use in the INCLUDE.

When the INCLUDE action is selected, **XLT File** lists all translation files except the current xlt file. xlt files which do not have the same message formats as the current xlt file are also excluded. xlt files in the master (primary) site are included.

An xlt file with another format cannot be selected as a sub-translation file. A prompt is given if an xlt file with another format is selected.
- 3 Define the prefixes in the Source and Destination fields.

These replace the group prefixes in sub-translation with the specified ones in the current translation configuration.

Prefixes support multi-replacement pairs.

Adding a replacement pair in this table creates a new empty row in the Source panel.
- 4 Click **Apply** to apply this change. The included translation file is shown in the INCLUDE action, and all actions in the included translation file are expanded accordingly.

The host xlate shows the substituted addresses after applying the replacements in the action list. The actions included from a translation file cannot be edited.
- 5 Click the INCLUDE action button to expand or to collapse the included actions.

The engine reports an error if there is recursion in the included `xlt`.

- 6 There is a **Reload** context menu item for the INCLUDE action to reload the sub-translation file. This is enabled only for the INCLUDE action.
- 7 Save the current `xlt`. This creates a new OP named INCLUDE in the `xlt` file

XLATE debugger

There is no difference for the actions included from other `xlt` files for Xlate Debugger.

You can still set the breakpoint on the actions included from other `xlt` files and do the debugger normally.

Breakpoints cannot be set on the INCLUDE action line, similar to COMMENT/ELSE.

The engine loads the included `xlt` file in runtime. For the Xlate Debugger, the GUI keeps the same action index number with the engine.

The engine ignores the INCLUDE action when doing the debugger for the next action and step in the included actions.

Information storage

When the `xlt` file is saved, an OP INCLUDE and parameters FILE, REPLACE_IN, REPLACE_OUT, IN and OUT are added.

FILE specifies the sub-translation file.

IN/OUT specifies the Source/Destination prefixes in the host `xlt` file.

REPLACE_IN/REPLACE_OUT specify the Source/Destination prefixes in sub-translation file. In Runtime, the engine replaces the prefix specified by REPLACE_IN/REPLACE_OUT with the one specified by IN/OUT for Source/Destination.

For example, the sub-translation file (`pid.xlt`) contains:

```
{ { OP COPY }
  { ERR 0 }
  { IN 0(0).PID(@s).#1(0) }
  { OUT Field_Field }
}
{ { OP COPY }
  { ERR 0 }
  { IN 0(0).PID(@s).#5(0) }
  { OUT Field_Field1 }
}
{ { OP COPY }
  { ERR 0 }
  { IN 0(0).PID(@s).#8(0) }
  { OUT Field_Field2 }
}
```

The main `xlt` file contains:

```
{ { OP ITERATE }
  { BASIS 0(0).PID }
  { VAR %s }
  { TYPE segment }
  { BODY {
    { { OP COMMENT }
```

```

        { COMMENT {TODO: Insert new actions here} }
    }
    { { OP COPY }
      { ERR 0 }
      { IN {{$s}} }
      { OUT =@s }
    }
    { { OP INCLUDE }
      { FILE pid.xlt }
      { REPLACE_IN {} }
      { REPLACE_OUT {} }
      { IN {} }
      { OUT {} }
    }
  }
}

```

MATH

This action performs the specified math operation on the source values. Specify at least two source values for a math operation. One or more can be a constant value.

Select the math function from the **Function** list:

- **ADD** adds the source values. The results are sent to the location that is specified in the Destination pane.
- **AVG** takes the average of the source values. The results are sent to the location that is specified in the Destination pane.
- **DIV** divides the first source value by the second source value. The results are sent to the location that is specified in the Destination pane.
- **MAX** takes the maximum of the source values. The results are sent to the location that is specified in the Destination pane.
- **MIN** takes the minimum of the source values. The results are sent to the location that is specified in the Destination pane.
- **MOD** takes the remainder after dividing the first source value by the second source value. The results are sent to the location that is specified in the Destination pane.
- **MUL** multiplies the source values. The results are sent to the location that is specified in the Destination pane.
- **SUB** subtracts the first source value from the second source value. The results are sent to the location that is specified in the Destination pane.

STRING

This action performs the specified string operation on the source value, and the result is saved to the location that is specified in the destination pane. Several STRING functions have the parameter when in action.

The STRING action provides string operations. This action is user-extendable. This action calls a predefined Tcl procedure. The predefined Tcl procs are related to the STRING operation.

The Content area is used to present the statement to invoke the Tcl procedure. Double-clicking a Function entry appends the usage into the Content. After this, you edit the content of the statement according to the actual requirement.

Pausing the mouse on a function shows its usage. Double-clicking the function places it into the Content area.

This table shows the available STRING operations:

Operation	Description
<code>xlateStrConcat</code>	Concatenates the source values as a string. Multiple source values are supported.
<code>xlateStrIndex charIndex</code>	Gets the string indicated by the index that is specified by the function parameter from the source value.
<code>xlateStrLength</code>	Gets the length of the source value as the string.
<code>xlateStrMap ?nocase? mapping</code>	Replaces the substring in the source value based on key-value pairs in <code>mapping</code> . <code>mapping</code> is a key value Tcl-style list. If <code>-nocase</code> is specified, then matching is performed without regard to case differences.
<code>xlateStrPadLeft/Right length ?padchar?</code>	Pads the source value from left/right side with the character specified by <code>padchar</code> to the length. If <code>padchar</code> is not specified, then whitespace is taken to fill.
<code>xlateStrReplace first last ?newstring?</code>	Removes a range of consecutive characters from the source value. This starts with the character whose index is <code>first</code> and ends with the character whose index is <code>last</code> . An index of 0 refers to the first character of the string. If <code>newstring</code> is specified, then it is placed in the removed character range. If <code>first</code> is greater than <code>last</code> or the length of the initial string, then the result is the same as the source value. This also happens when <code>last</code> is less than 0.
<code>xlateStrSplit ?splitchar?</code>	Splits the source value into several substrings by the separator that is specified by <code>splitchar</code> . The results are sequentially saved to the location that is specified by the destination pane. If <code>splitchar</code> is not specified, then whitespace is taken by default.

Operation	Description
<code>xlateStrSubstring first last</code>	<p>Gets a range of consecutive characters from the source value. This starts with the character whose index is <code>first</code> and ends with the character whose index is <code>last</code>. An index of "0" refers to the first character of the string.</p> <p>If <code>first</code> is greater than <code>last</code>, then an empty string is returned.</p> <p>You can specify <code>end</code> to reach the end of the string. If <code>first</code> and <code>last</code> are not specified, then the range is from "0" to the end of the string.</p>
<code>xlateStrToLowercase/xlateStrToUppercase</code>	<p>Applies lowercase/uppercase to the source value, and saves the result to the location that is specified by the destination pane.</p>
<code>xlateStrTrim ?chars?</code>	<p>Gets a value equal to the source value, except that any leading or trailing characters present in the string given by <code>chars</code> are removed.</p> <p>If <code>chars</code> is not specified, then white space is removed. This includes spaces, tabs, newlines, and carriage returns.</p>
<code>xlateTrimLeft/xlateStrTrimRight ?chars?</code>	<p>Gets a value equal to the source value, except that any leading/trailing characters present in the string given by <code>chars</code> are removed.</p> <p>If <code>chars</code> is not specified, then white space is removed (spaces, tabs, newlines, and carriage returns).</p>

COMMENT

This action provides a means to include a text comment with the translation definition.

In the Comment pane, specify any text comment to include with the translation definition in this pane. Comments are often required for complex or translation actions.

Entered comments are maintained in the `xlt` file and can be edited in later updates of the Translation Configurator.

BULKCOPY

This action copies every common input record field to an output record field, applying all field width and repetition count variations.

Any fields that happen in the input record that do not exist in the output record are not copied.

BULKCOPY provides for automatic copying between two variants of UN/EDIFACT, HL7, XML, or X12.

This action is effective for copying from an HL7 to HL7, UN/EDIFACT to UN/EDIFACT, or X12 to X12.

PATHCOPY

This action moves and clears information between messages with more control than that provided by BULKCOPY. It does not resort to the detail that is required by COPY.

This action expands upon the BULKCOPY and COPY actions. Any unit of hierarchical organization, for example, group, segment, field, and so on, can be addressed in one action statement.

Basically, BULKCOPY copies an entire message format. PATHCOPY copies sections of a message format.

Note: PATHCOPY, by design, only supports copying "input record" to "output record," not "output record" to "input record."

Data bundles of arbitrary size

PATHCOPY is closely related to the hierarchical data formats such as HL7, UN/EDIFACT, NCPDP, and X12. PATHCOPY also adds the ability to move and relocate fields for FRL and VRL message formats.

PATHCOPY can replicate a datum into fields and subfields for HRL, FRL, and VRL messages.

Points to remember

- With PATHCOPY, you can use partial- or full-path specifiers as parameters. These parameters designate how data is moved between IN, input record messages, and OUT, output record messages.
 - Full-path specifiers to a field can be used for both input records and output records. The result is the same as having used a standard COPY operation to perform the move of the field or component.
 - A full-path specifier that goes to a single atomic datum, or the specification of a constant such as null, can be used in the input record. That datum or constant is replicated into each location for data elements. This is indicated by the output record path designator.
 - If partial-path specifiers are used for both input records and output records, then they must designate similar data entities. For example, PID segment to PID segment.
- Partial path for HL7, UN/EDIFACT, NCPDP, and X12 is any path that goes to at least a group. For example, 0, 0(1), 0(0).PID, 0(0).PID(1). Use BULKCOPY for full message to full message copy for all messages. PATHCOPY of a group to a group can be equivalently used in some instances.
- Partial path for FRL and VRL is a path to a field.
- Full path for HL7, UN/EDIFACT, NCPDP, and X12 is any path that goes to at least a field.
- Full path to atomic datum is any path that resolves to an individual data element.
 - In HL7, UN/EDIFACT, NCPDP, and X12, this is a path to a single subcomponent.
 - In VRL, this is usually a path to a single sub-subcomponent.
 - In FRL, this is usually a path to a single subcomponent.
- With PATHCOPY, you can relocate a data entity. For example, you can move a segment from one position of an input record to another position for an entity within the output record hierarchy. For example, a segment may be moved between nested groups in HL7.

- PATHCOPY follows the established behaviors of BULKCOPY and COPY for moving data between HL7, UN/EDIFACT, NCPDP, and X12 message variants.
- Multiple path designators can be entered into **Source** and **Destination** of the GUI. They can also be entered into the IN, input record, and OUT, output record, keyed lists of the PATHCOPY scripted configuration file. The sense of these multiple entries is that of pairing, where the first Source argument applies to the first Destination argument. Extraneous, unpaired entries are ignored.
- PATHCOPY can be used within and in conjunction with ITERATE translation blocks. The designation of entities to be iterated within a PATHCOPY path parameter is the same as that used for other actions such as COPY.
- PATHCOPY can be used for automatic, implied iteration copies. To perform automatic iteration, leave an explicit repetition designator off of the end of the input record or output record partial path.
- PATHCOPY can be used within and in conjunction with IF-ELSE translation blocks.
- PATHCOPY can be used to move data between related formats, such as FRL/VRL or HL7/X12. Movement of more than atomic data elements between unlike formats, such as FRL/HL7, is undefined.

Pathcopy usage

The PATHCOPY action expands upon the BULKCOPY and COPY actions. Any unit of hierarchical organization (group, segment, field, and so on) can be addressed in one action statement.

PATHCOPY is used in these ways:

- To replicate a unit of hierarchical organization from an input message into any appropriate location for that unit in the output message:
 - To use PATHCOPY, the partial path specifications of the input and output must match in organizational type. For example, you can only PATHCOPY an HL7 PID to an HL7 PID. This use of PATHCOPY operates across the full spectrum of organization, from replicating a group to a group, down to subcomponent to subcomponent. In this respect, PATHCOPY can be used in many situations as a replacement for BULKCOPY and COPY actions.

Format versions can differ between the input and output. For example, you can PATHCOPY between an HL7 v2.1 input stream and an HL7 v2.3 output stream.
- To replicate one datum into all of the data locations within a large unit of hierarchical organization (group, segment, field, and so on):
 - This behavior of PATHCOPY is triggered by specifying a single datum as the input to the action. The one datum can be from an input message. It can also be a temporary variable constant, for example, @null, or it can be a user-supplied string constant, for example, =a_string. If the datum is from an input message, then the path pointing to that message datum must be fully-qualified to the sub-component level.
 - This function of PATHCOPY is useful to clear out and remove unnecessary portions of an output message. For example, an unnecessary segment that was copied to the output message by a BULKCOPY action. Use the @null temporary variable as the PATHCOPY input to clear and remove the unnecessary portions.

Note: If, within the same message formats, you do a one-to-many PATHCOPY, then the outbound message structure is not created. In the same message formats, PATHCOPY can be the first operation in the translation. When you do a one-to-many PATHCOPY, the outbound message structure is not created. For example, =mys tring--> 0(0).1(0).PID" [HL7-->HL7]. If the structure is already there, then it overwrites it. For example, a BULKCOPY is followed by @null copied to all of the PID segments.

PATHCOPY, by design, only supports copying "input record to output record", not "output record input record."

CALL

This action calls a Tcl procedure to perform a set of operations on the source values.

This action provides direct access to Tcl for custom operations not directly supported by the translation system. This gives complete control over the translation action by specifying a Tcl code segment to be run instead of one of the standard actions. A specific API is provided for interacting with the translation system.

In the Proc pane, Tcl and Java procedures use the same text box.

When **Tcl** is selected, click the button to open the **Tcl Script Editor**.

When **Java** is selected, click the button to open the **Java Procedures Settings** dialog box.

CONTINUE

This action generates a message from all the translated elements up to the CONTINUE, and places it on the disposition list as CONTINUE. This includes further processing of the generated message.

CONTINUE lets the generated message continue through normal processing.

DATECOPYOPT

This action defines how to handle date copies. You can also specify the century to use when it is not present in the data.

For **Date Delimiter**, click the arrow to select the character to use when separating the elements of a date value in an outbound message.

For **Time Delimiter**, click the arrow to select the character to use when separating the elements of a time value in an outbound message.

For **Add Precision**, select the check box for all optional fields to be in the outbound message for the Date/Time object. Clear the check box for optional fields to be in the outbound message only when they are present in inbound data.

Missing Centuries pane

For **Use System Date**, select this to use the century of the current date given by the host operating system as the century on the outgoing date. This is used when an incoming date is missing a century.

For **Use Date Rule**, select this to use the rule specified by the associated options for determining the century of the outgoing date. This is used when an incoming date is missing a century. Click the arrows next to the parameter to select a value.

Time Options pane

XML messages have a `DateTime` type. The value consists of date, time, and timezone.

In the previous versions, the Xlate action `DATECOPYOPT` in the Cloverleaf engine would make `DateTime` be converted into the local timezone during the translation. The date and time would be changed according to the new timezone. There was no method of disabling this timezone conversion. (This was fixed in CIS 6.1.3.2).

Select **Use local timezone** to apply the local time zone of your machine to the input date time in the translation of `DATECOPYOPT`. This is useful when copying the date/time from input to output. By default, this is selected. If the value is configured with "0", then the timezone conversion is disabled and `DateTime` follows the timezone in the input date time. This indicates the `DateTime` string looks similar as before, after the translation.

The corresponding key in the `DATECOPYOPT` action is `USELOCALTZ`.

When this option is cleared, the time zone conversion is disabled, and the date time follows the time zone in the input date time.

For backward compatibility, this option is selected by default. This indicates the Cloverleaf engine always converts the time zone into the local time zone date in the translation.

Select **Use system time** to reference the system time when filling in missing values in the incoming date. This is used if an incoming date is being copied to an outgoing date/time value.

Selecting **Use custom time** specifies the values to use to fill in any missing parameters in the incoming date. This is used if an incoming date is being copied to an outgoing date/time value. Click the arrows next to the parameter to configure the custom value.

Default Timestamp Precision

On the **Client Preferences > Xlate Config** tab is the **Default Timestamp Precision** option.

Select this to set the default value of the **Add Precision** property of the `DATECOPYOPT` action in Translation Configurator.

When you select **On**, the **Add Precision** check box on the `DATECOPYOPT` action dialog box is selected.

When you select **Off**, the default, the **Add Precision** check box on the `DATECOPYOPT` action dialog box is cleared.

IF

This action provides controls for defining a condition and the action to be taken if that condition exists.

In the Operators pane, **Category** lists the available String, Numeric, Logical, and Other operators. Selecting a category lists the corresponding operators and descriptions in the adjoining text box.

After selecting the operators, click the arrow to add them to Condition, next to IF. You can also double-click an operator to populate Condition.

In the Condition pane, IF accommodates one or more lines of text. Scrollbars display if more text is added than what can be viewed.

Conditional operations are specified by IF/ELSE in the Translation Configurator.

Selecting an IF action provides controls for defining the condition and the action to be taken if that condition exists. An optional ELSE action is also provided.

Select **Activate Optional 'ELSE'** to create an Else action.

Conditional operations

Conditional operations are specified by IF/ELSE in the Translation Configurator.

Selecting an IF action provides controls for defining the condition and the action to be taken if that condition exists. An optional ELSE action is also provided.

Options pane

The **Condition** field can accommodate one or more lines of text. Scrollbars are displayed if more text is added than what can be viewed.

The Operators list boxes show the available lists of comparison, logical, and precedence operators for the selected **Category**.

Choosing a **Category** and an operator and then clicking the arrow adds it to the **Condition** field.

If no text is selected, then the operator is inserted at the cursor.

Operators may also be added to the text box by selecting the operator and double-clicking.

A complete conditional expression contains at least one IF action. The ELSE action is optional. Conditional expressions can be nested to as many levels as necessary. They can also be nested within ITERATE actions, and ITERATE actions can in turn be nested within conditionals.

If an operation keyword is added to the Translation Configurator without an action, then the keyword is ignored during processing. For example, IF alone is moved into a translation sequence.

The condition that is specified in an IF/ELSE operation can use any combination of these data references:

- Input fields, including iteration variables
- Output fields (~)
- Temporary fields (@)
- String constants (=)
- Numeric constants (=)

Operators

This table shows the available operators for various categories:

Category	Available operators
Text comparisons	<ul style="list-style-type: none"> lt: less than le: less than or equal to eq: equal to ne: not equal to ge: greater than or equal to gt: greater than ct: contains <p>Text operators perform string comparisons, no matter what data types are specified in the arguments.</p> <p>If an operand is a date type, then both operands are converted to dates and the dates are compared.</p>
Numeric comparisons	<ul style="list-style-type: none"> < : less than <=: less than or equal to ==: equal to !=: not equal to >=: greater than or equal to >: greater than <p>Numeric operators expect their arguments to be numeric. Use numeric operators to compare any combination of numeric data types. When different types are compared, operands are promoted to the required type.</p> <p>If both operands are integers, then an integer comparison is performed. If an operand is a floating point, then a double comparison is performed.</p>
Logical comparisons	<ul style="list-style-type: none"> !:NOT : OR &&: AND <p>Comparisons can be as complex as necessary and can use parentheses to define the precedence of operations.</p>

Category	Available operators
Other comparisons	<p>The only option is <code>[()]</code> Evaluate First.</p> <p>This binds compound boolean expressions, so that the evaluation happens correctly. For example:</p> <pre>IF (Flag eq =A Flag eq =B) && Dept eq =L</pre> <p>In this example, if <code>Flag</code> equals A or B, and <code>Dept</code> is also L, then the whole statement is true.</p> <p>Note: Conditionals are not validated in any way.</p>

SEND

This action generates a message from all the translated elements up to the `SEND`, and transmits that message to the outbound connection.

The `SEND` action bypasses any post-processing. Otherwise, it is the same as the `CONTINUE` action.

SUPPRESS

This action transmits the message. It is the last translation operation if `SEND` or `CONTINUE` actions are used to generate multiple outbound messages from one inbound message. For example, `CONTINUE`, then `SEND`, then `SUPPRESS`.

`SUPPRESS` suppresses processing of the original input message beyond the XLT phase.

BREAK

`BREAK` is used to break a loop or to skip `ITERATE`s.

- If `BREAK` has an earlier `SUPPRESS`, then no message is sent. If there is no earlier `SUPPRESS`, then it sends what it has at that point, which could be a partial message.
- If `BREAK` is inside an `ITERATE`, then it breaks the iteration.
- If `BREAK` is outside of `ITERATE`, then it stops the translation completely.
- If `BREAK` is in an `IF` action on the root level, then the engine exits the entire translation.

In an `ITERATE`, the `BREAK` action causes the translation engine to immediately exit the current `ITERATE` and move to the next action after the `ITERATE`. If the `ITERATE` is nested, then the `BREAK` only breaks out of the current loop, not the entire nest.

If a `BREAK` is encountered outside of an `ITERATE`, then the translation ends immediately and the current parse tree is encoded. Rules set by `SUPPRESS` are followed as if the translation had normally ended.

Actions on the same level and behind the `BREAK` are unreachable.

To use BREAK on a condition in an ITERATE, use IF and BREAK together.

Note: BREAK is restricted to ITERATE.

Using BREAK

- 1 Select a row on the configuration table.
- 2 Click **BREAK**. A new BREAK row is added after the selected row.
The BREAK action is similar to CONTINUE, and does not have any parameters.
- 3 Save the configuration. This writes the BREAK action to the `xlt` file.

Inserting a BREAK in an IF action

- 1 Define an IF action at the root level.
- 2 Define a BREAK action under the IF action.
- 3 Define additional actions under the BREAK.
Note: These actions do not run if the BREAK is triggered.
- 4 Test the xlate file in `hcixlttest`. When BREAK is triggered, the engine exits the translation.

Action properties

Translation actions are tools that the Translation Configurator applies during the translation process. These actions convert the contents of the source message from its original format to the destination message format. Translation actions can copy the contents of one field verbatim to another. They can also apply multi-step algorithms that change both the arrangement and contents of message components.

In addition to actions that apply to translation processing, Tcl procedures can be invoked to perform pre-translation and post-translation processing. For example, you can apply extended expressions to transform the contents of input message components before copying or concatenating.

Similarly, you can use Tcl procedures to examine messages after translation, but before transmission, to create customized translation activity reports.

Temporary variables are accessed by right-clicking within the relevant text-entry components and selecting from the menu.

Action properties pane

Use the Action Properties pane to view or edit actions. When editing, the Action Properties pane is mode-less, giving access to other dialog boxes.

The action properties pane is visible only when an action is selected. When the Translation Configurator is initially opened, by default there is no action and the Action Properties pane is not shown. When a translation file is initially opened, by default there is no action selected and the Action Properties pane is not shown.

Clicking **Toggle Input Format** and **Toggle Output Format** opens the Input Message Format and Output Message Format panels on the side of the Action Properties pane.

These panels are active and you can double-click a path or use **Add to Source** or **Add to Destination** to insert a path into the action.

To view the properties of an action, select the action for the Action Properties pane to show its properties.

When a new action is created, a new action row is added to the action table. It is highlighted to show the properties of the new action. The highlighted action's description is updated when properties are changed.

Any inputs to Source and Destination update the Source/Destination column of the highlighted action.

Changes to the pre- or post-procs text box are immediately shown in the pre/post column.

Selecting another action in the action combo box updates the Operator column of the highlighted action.

Multiple copied actions

You can paste multiple copied actions. If those copied actions are in different levels, then they are pasted in the top level.

Note: ELSE cannot be separately copied without the IF action. If you select ELSE without IF and right-click, then the **Copy** menu item is not available.

Apply action properties

You can apply properties from the **Edit** menu using **Ctrl+Shift+A**, right-clicking in the Operation List pane, or clicking **Apply** next to the **Error** menu.

Enable or disable auto-apply from the **Client Preferences > Xlate Config** tab by clearing/selecting **Automatically apply action properties change**.

When auto-apply is enabled, **Apply** is not available. All modifications are automatically applied.

By default, the auto-apply setting is disabled.

When auto-apply is disabled, you must manually apply the modification by clicking **Apply**.

After you apply the modification, the translation is then marked as modified. If you switch to another action without clicking **Apply**, then a message box opens asking for confirmation of the modification.

If you do not apply the modification, then all modifications are discarded and the action property is restored to its last value.

Expand operation statements

To expand/collapse operation statements, select **Automatically expand operation statement upon entry** on the **Client Preferences > Xlate Config** tab. This automatically expands all multi-level and nested IF and ITERATE (" +/-") statements.

Buttons are also located on the New Action toolbar that can be used to expand or collapse all "+/-" operation statements. File menu options are also available.

Translation Configurator examples

The Translation Configurator provides quick access to the most frequently used translation actions.

These examples introduce how to perform various functions, including:

- Using ITERATE to perform the same translation actions over many repeating elements in a message.
- Defining a record translation from one format to another.
- Using translation tables to convert input data to output data.
- Using a math operator to combine two or more input data fields into a single output value.
- Using PATHCOPY to create XML-to-XML translations.

Procedure calls

The Translation Configurator provides quick access to the most frequently used translation actions, but it is not exhaustive. For complex translations, write translation algorithms and procedures using the Tcl extension language.

The **Pre Procs** and **Post Procs** processor fields are available with several translation operations. You can write Tcl code directly and incorporate it into the translation file.

Test your translation configuration before putting it into use. Click **Test** to open the Testing Tool to test the configuration before putting it into production.

Specifying the translation action

- 1 Determine where the action is placed by selecting an action mode.
- 2 Select an action from the New Action toolbar.
- 3 Select the **Pre Procs** or **Post Procs** tab and select **Tcl** or **Java** to open the **Script Editor**.
- 4 Specify Tcl code in the **Script Editor**.

Specifying the source

A source value can come from the input record, the output record, if it already contains the data, a temporary data field, or a constant. In this case, the Source value comes from the input record.

- 1 Click **Toggle Input Format** on the Action Properties pane. The **Input Message Format** panel is displayed.
- 2 Click the plus sign next to the field name to view the subfield.
- 3 Select the subfield to use for the source value.
- 4 Click **Add to Source**.

Specifying the destination

A destination value can come from a temporary field or the output record. In this case, the Destination value comes from the output record.

- 1 Click **Toggle Output Format** on the Action Properties pane. The **Output Message Format** panel is displayed.
- 2 Click the plus sign (+) next to the field name to view the subfield.
- 3 Select the subfield to use for the destination value.
- 4 Click **Add to Destination**.
- 5 Save the file.

Iterative operations

Use ITERATE to perform the same translation actions over many repeating elements in a message. This creates a loop in the translation. For example, it copies a repeating field from the input record to a corresponding field in the output record.

Configure each operation separately, or configure one operation with a loop around it to repeat the appropriate number of times. This section shows the latter.

Specifying the ITERATE translation action

Begin the translation definition by configuring the translation operation.

- 1 Determine where the action is placed by selecting an action mode.
- 2 Select **ITERATE** on the **New Action** toolbar to place it on the list.
- 3 For **Type**, arrow and select the type of record element to use as the basis for the loop. The default is **list**.
- 4 Click **Toggle Input Format** or **Toggle Output Format** in the Action Properties pane. This opens the selected panel.
- 5 Click the plus sign next to the field name to view the subfields.
- 6 Select the subfield to use as the basis.
- 7 Click **Copy to Basis**.
- 8 Specify the iteration counting **Variable** to use. All variables must start with % and be "f" (field), "g" (group), "l" (list), or "s" (segment). With multiple variables, for example three field variables, use %f1, %f2, and %f3.
- 9 Click **Apply**.
- 10 Click **OK**. The Translation Configurator displays showing the new configuration.

- 11 Click the plus sign next to the new **Iterate** action.
- 12 Click **Comment** to highlight it.
- 13 Select the action that forms the body of the loop. A loop can contain any number of translation actions. For the input and output values of the operation, use the iteration counter to specify the repetition of the field.
- 14 Click **Apply**.
- 15 Click **OK**.
- 16 Save the file.
- 17 Click **Test** to display the Testing Tool to test the configuration before putting it into production.

Configuring a DATECOPY action

- 1 Create an action by clicking **DATECOPY**.
- 2 Select a **Date Delimiter** from the . The default is “/”.
- 3 Select a **Time Delimiter** from the . The default is “:”.
- 4 Configure the Missing Centuries pane.
There are two options: **Use system date** (default) and **Use date rule**.
When **Use system date** is selected, all date fields are unavailable.
- 5 Configure the Time Options pane.
There are two options: **Use system time** (default) and **Use custom time**.
When **Use system time** is selected, all fields are unavailable.
- 6 Save the file.

Configuring IF/ELSE, CONCAT and CONTINUE actions

- 1 Create a new action by clicking **IF**.
- 2 Select **Pad** from the **Error** menu.
- 3 Double-click the Input Message.
This inserts it into the Condition pane.
- 4 On the Operators pane is a **Category** menu which contains four items: STRING, NUMERIC, LOGICAL and OTHER. When an item from the menu is selected, the text field displays the available operators.
Select an operator **Category** and from the list double-click and operator to place it into the Condition pane. You can also select an operator and click the arrow.
- 5 Specify a constant in the Condition pane (for example, =Kaiser).
Click **Apply**.
- 6 Add a **CONCAT** action, and select the Source values from the Input Message Format.
- 7 Select the Destination values from the Output Message Format.
- 8 On the IF action, select the **Activate Optional 'ELSE'** check box and click **Apply**.
- 9 Add the **CONTINUE** action.

- 10 Click **Apply** and save the file.

Selecting record layouts for the input and output records

- 1 Click **New** to open the **Choose File Formats** dialog box.
Incoming and outgoing record file formats must be predefined in the configurator for each type. These predefined formats are saved with an extension denoting its type. For example, `.frl`, `.vrl`, and so on. This is the file to which the **Choose File Formats** dialog box refers.
- 2 Select the input **Format** in the Input Record Format pane. Configure the remainder of the input format.
- 3 Select the output **Format** in the Output Record Format pane. Configure the remainder of the output format.
- 4 Select the **Insert an Existing XLT** check box to insert another translation file, if required.
Click the button to open the **File Browser** dialog box. Use this dialog box to select from existing translation files.
- 5 Click **OK**.

Format-to-format conversion

These examples show how to define a record translation from one format to another.

Begin the translation specification by selecting the record layouts for the input and output records. Then configure the individual translation actions.

Specifying the translation action

- 1 Determine where the action is placed by selecting an action mode.
- 2 Select an action from the New Action toolbar.
- 3 Select the **Pre Procs** or **Post Procs** tab and select **Tcl** or **Java** to open the **Script Editor**.
- 4 Specify Tcl code in the **Script Editor**.

Specifying the source

A source value can come from the input record, the output record, if it already contains the data, a temporary data field, or a constant. In this case, the Source value comes from the input record.

- 1 Click **Toggle Input Format** on the Action Properties pane. The **Input Message Format** panel is displayed.
- 2 Click the plus sign next to the field name to view the subfield.
- 3 Select the subfield to use for the source value.
- 4 Click **Add to Source**.

Specifying the destination

A destination value can come from a temporary field or the output record. In this case, the Destination value comes from the output record.

- 1 Click **Toggle Output Format** on the Action Properties pane. The **Output Message Format** panel is displayed.
- 2 Click the plus sign (+) next to the field name to view the subfield.
- 3 Select the subfield to use for the destination value.
- 4 Click **Add to Destination**.
- 5 Save the file.

Saving the translation action

- 1 Click **Save** on the IDE toolbar to save the translation action to a file. This opens a file selection dialog box.
- 2 Specify the file name or select a file name from the list.
- 3 Click **Save**.
- 4 Test the configuration before putting it into production.

Translation tables

Translation tables convert input data to output data. One use for a translation table is to convert a code value from the input record into an expanded text field for the output record.

These procedures show how to configure a translation using a translation table for the account code.

Specifying the TABLE translation action

- 1 Select **TABLE** on the New Action toolbar.
- 2 Open the **Table** menu in the lower pane and select from the list of available translation tables.
- 3 If required, then on the **Side** menu specify which side of the table to use for looking up input data values. This is usually the left side of the table, which is named Input.
Note: The TABLE translation action does not default to a table, and the selection box is editable. You can create translation templates and save them for later use.

Specifying the TABLE source value

A source value can come from the input record, the output record, if it already contains the data, a temporary data field, or a constant. In this case, the Source value comes from the input record.

- 1 Click **Toggle Input Format** on the Action Properties pane. The **Input Message Format** panel is displayed.
- 2 Click the plus sign next to the field name to view the subfield.

- 3 Select the subfield to use for the source value.
- 4 Double-click or click **Add to Source**.

Specifying the TABLE destination value

Destination values can come from a temporary field or a component of the output record. In this case, the Destination value comes from the output record.

- 1 Click **Toggle Output Format** on the **Action Properties** pane. The **Output Message Format** panel is displayed.
- 2 Click the plus sign next to the field name to view the subfield.
- 3 Select the subfield to use for the destination value.
- 4 Double-click or click **Add to Destination**.
- 5 **Save** the translation file.
- 6 Test the configuration before putting it into production.

Configuring the procedure call action

- 1 On the **Pre Procs** tab, select the **Tcl** option.
- 2 To create your own Tcl procedure, click **Tcl Script Editor**. The **Tcl Script Editor** dialog box is displayed.
- 3 To import existing Tcl procedures, right-click in the text box and select **Insert**.
- 4 Select a procedure from the **Proc** menu and click **OK**.
- 5 On the **Pre Procs** tab, select the **Java** option. You can input your own Java code in this text box or select existing Java code.
- 6 To import existing Java code, right-click in the text box and select **Edit**. The **Java Pre-Procedure Settings** dialog box is displayed.
- 7 Specify a **Class**.
- 8 Click **OK**. This imports Java code into the text field.

Math operations

Use a math operator to combine two or more input data fields into a single output value. The math operator is applied to each input value in the order that the value displays in the source (input) list.

Math operations result in integers. The division (DIV) and average (AVG) operators use real, floating point, numbers for their intermediate calculations. The final result is rounded up for fractional values of .5 and above, and rounded down for fractional values below .5.

When converting to HEX types, the correct decimal to hex is performed. Conversely, when converting from HEX types, the value is properly scanned as a hex number.

Specifying the MATH translation action

- 1 Select **MATH** to place it on the list.
- 2 Open the **Function** menu of available math functions and select a function.

Specifying the MATH source value

Destination values can come from a temporary field or the output record. In this case, the Destination value comes from the output record.

- 1 Click **Toggle Input Format** on the Action Properties pane. The **Input Message Format** panel is displayed.
- 2 Click the plus sign (+) next to the field name to view the subfield.
- 3 Select the subfield to use for the source value.
- 4 Click **Add to Source**.

Specify the MATH destination value

Destination values can come from a temporary field or the output record. In this case, the Destination value comes from the output record.

- 1 Click **Toggle Output Format** on the Action Properties pane. The **Output Message Format** pane is displayed.
- 2 Click the plus sign (+) next to the field name to view the subfield.
- 3 Select the subfield to use for the destination value.
- 4 Click **Add to Destination**.
- 5 **Save** the file.

Configuring the math action

- 1 On Input Message Format pane, select the subfields to add and click **Add to Source**.
- 2 On the Output Message Format pane, select the destination subfield and click **Add to Destination**.
- 3 Click **Apply** and save the file.
- 4 Test the configuration before putting it into production.

Calling the PATHCOPY function

An additional feature of PATHCOPY interprets partial path specifications that do not end in an explicit repetition as an instruction to automatically iterate.

For example, the input and output paths do not end in target repetitions. The output message contains iterated components. PATHCOPY automatically moves all the repetitions of the partial path from the input message to the output message.

In another example, the output partial path does not target an explicit repetition. The output message contains iterated components. PATHCOPY fills all the datum locations in all iterations of the partial path in the output message with the input datum.

Example message

This message is used for the remainder of the examples:

```
MESSAGE 1
.
.
0(0).GT1(0).00405(0) : >154<
0(0).GT1(0).00406(0) : >155^^^i676-<
0(0).GT1(0).00407(0) : >j677-----^k678-----^l6<
0(0).GT1(1).00405(0) : >172<
0(0).GT1(1).00406(0) : >173^^^F698-<
0(0).GT1(1).00407(0) : >G699-----^H700-----^I7<
0(0).GT1(2).00405(0) : >190<
0(0).GT1(2).00406(0) : >191^^^c720-<
0(0).GT1(2).00407(0) : >d721-----^e722-----^f7<
0(0).GT1(3).00405(0) : >208<
0(0).GT1(3).00406(0) : >209^^^z742-<
0(0).GT1(3).00407(0) : >A743-----^B744-----^C7<
0(0).GT1(4).00405(0) : >226<
0(0).GT1(4).00406(0) : >227^^^W764-<
0(0).GT1(4).00407(0) : >X765-----^Y766-----^Z7<
.
.
.
```

Example 1: PATHCOPY from 0(0).GT1(0) to 2(0).GT1(0)

```
MESSAGE 1
.
.
2(0).GT1(0) : >|154|155^^^i676-|j677-----^k678-----^l6<
.
.
.
```

Example 2: PATHCOPY from 0(0).GT1(0) to 2(0).GT1

```
MESSAGE 1
[0:TEST] Illegal pathcopy from "0(0).GT1(0)" (segment) to "2(0).GT1" (segment with automatic iteration).
```

This fails because there is a hierarchy level mismatch: the source specifies Segment. The destination specifies Segment with Automatic Iteration.

Example 3: PATHCOPY from 0(0).GT1 to 2(0).GT1(0)

```
MESSAGE 1
[0:TEST] Illegal pathcopy from "0(0).GT1" (segment with automatic iteration) to "2(0).GT1(0)" (segment).
```

This fails because there is a hierarchy level mismatch: the source specifies Segment. The destination specifies Segment with Automatic Iteration.

Example 4: PATHCOPY from 0(0).GT1 to 2(0).GT1

```

MESSAGE 1
.
.
2(0).GT1(0) : >|154|155^^^i676-|j677-----^k678-----^l6<
2(0).GT1(1) : >|172|173^^^F698-|G699-----^H700-----^I7<
2(0).GT1(2) : >|190|191^^^c720-|d721-----^e722-----^f7<
2(0).GT1(3) : >|208|209^^^z742-|A743-----^B744-----^C7<
2(0).GT1(4) : >|226|227^^^W764-|X765-----^Y766-----^Z7<

```

Example 5: PATHCOPY from 0(0).GT1 to 2(0)

```

MESSAGE 1
.
.
2(0).GT1(0) : >|154|155^^^i676-|j677-----^k678-----^l6<
2(0).GT1(1) : >|172|173^^^F698-|G699-----^H700-----^I7<
2(0).GT1(2) : >|190|191^^^c720-|d721-----^e722-----^f7<
2(0).GT1(3) : >|208|209^^^z742-|A743-----^B744-----^C7<
2(0).GT1(4) : >|226|227^^^W764-|X765-----^Y766-----^Z7<
- Same as 0(0).GT1 to 2(0).GT1

```

Example 6: PATHCOPY from 0(0).GT1(0).#3(0) to 2(0).GT1(0).#3(0)

```

MESSAGE 1
.
.
2(0).GT1(0) : >|||j677-----^k678-----^l6<

```

There is inter-version conversion for 0(0).GT1(0).00405(0) (#1). The engine looks for the correct index of field 00405 under GT1 in the destination version.

Example 7: PATHCOPY from 0(0).GT1(0).#3 to 2(0).GT1(0).#3(0)

```

[0:TEST] Illegal pathcopy from "0(0).GT1(0).00407" (field with automatic iteration) to
"2(0).GT1(0).00407(0)" (field).
.
.

```

This fails because there is a hierarchy level mismatch: the source specifies Field with Automatic Iteration. The destination specifies Field.

Example 8: PATHCOPY from 0(0).GT1(0).#1 to 2(0).GT1(0).#1

```

MESSAGE 1
.
.
2(0).GT1(0) : >|154<
- Same as 0(0).GT1(0).#1 to 2(0).GT1(0)

```

No inter-version conversion for 0(0).GT1(0).00405(0) (#1).

XML PATHCOPY support

PATHCOPY can be used to create XML-to-XML translations.

XML PATHCOPY can accept as its source a partial path, complete path, temporary variable, or a constant. The destination can be a partial or complete path.

XML PATHCOPY also permits PATHCOPY for elements of the same type.

Single source PATHCOPY

If the source references a single value, then that value is copied to all elements in the destination tree. Examples of a single value are a complete path, temporary variable, or a constant. This updates the values of all `#text` nodes or attributes that are currently present in the destination sub-tree referenced by the destination path.

By copying `@null`, a sub-tree can be removed from the destination XML message. If the destination path specifies a repetition, then only that specific repetition is modified.

Note: A source path referencing a destination value cannot be used. That is, "~" at the beginning of the path. This is a limitation that XML shares with other message formats.

Single source PATHCOPY examples

Example 1

Message:

```
<A>
  <B>
    <X name="test attribute">test value</X>
    <Y>test value 2</Y>
    <Z>test value 3</Z>
  </B>
  <C>
    <Q>test value 4</Q>
  </C>
</A>
Pathcopy = foo-->A.B
Resulting Message:
<A>
  <B>
    <X name="foo">foo</X>
    <Y>foo</Y>
    <Z>foo</Z>
  </B>
  <C>
    <Q>test value 4</Q>
  </C>
</A>
```

Example 2

Message:

```
<A>
  <B>
    <X name="test attribute">test value</X>
    <Y>test value 2</Y>
    <Z>test value 3</Z>
  </B>
  <C>
    <Q>test value 4</Q>
  </C>
</A>
```

```

Pathcopy = @null-->A.B
Resulting Message:
<A>
  <C>
    <Q>test value 4</Q>
  </C>
</A>

```

Multiple source PATHCOPY

If the input is a partial path, then values are copied to the destination by matching a source element name to a destination element name. The elements to which the source and destination paths refer must have the same name.

If the source path cannot be resolved to a single value (a partial path), then a multiple source PATHCOPY is performed. This operation does not modify values that already exist in the destination message. It only adds new elements. PATHCOPY attempts to copy each element from the source sub-tree to the destination.

Both paths must specify a repetition, or not specify a repetition. You cannot have only one of the paths specify a repetition.

PATHCOPY addresses ending in different names

Users are given the option of a multiple source XML PATHCOPY of addresses ending in different names.

For example, this is permitted:

```

OCM 1:
A
  B
    C
      D
      E

OCM2:
X
  Y
    C
      D
      E

PATHCOPY A.B-->X.Y

```

Note: Elements under the sub-root must have the same name. In the above example, C, D, and E must have the same name. The sub-root names can be different. For example, B versus Y in the above example.

Source and destination with identical OCMs

If the source and destination content models are identical, then the resulting destination sub-tree is identical to the source. All elements are copied to the destination maintaining their relative position in the sub-tree.

Child elements are added to the destination. This applies when the content model for the destination contains matching element names to child elements that exist in the source.

The XML content model can contain anonymous nodes that do not refer to an actual element in the XML message. These anonymous nodes are not matched in the PATHCOPY operation; instead, their children are matched, unless the child is also anonymous.

Source and destination with different OCMs

When the content models are not the same, a matching algorithm is used to match the source elements with elements in the destination content model. The content models must be similar for the matching algorithm to work. A value that is retrieved from the source is copied to the destination if a compatible path can be identified in the destination content model. This path is constructed from the starting point of the PATHCOPY to the #text node.

Example: Source path

Source OCM	Destination OCM
A {1,1} ()	X {1,1} ()
B {1,1} (,)	B {1,1} (,)
C {1,1} ()	θ {1,-1} (,)
#text	C {1,1} ()
D {1,1} ()	#text
#text	D {1,1} ()
	#text

If the PATHCOPY is from A.B in the source to X.B in the destination, then the working paths to match are B.C.#text and B.D.#text. The path does not start from the root of the message, only from the root of the sub-tree referenced by the PATHCOPY source path.

For path compatibility, each component of the path must have a matching node in the destination OCM tree. For a path component to match, it must have the same name and be the same type, choice or sequence, as the corresponding source node. The permissible repetitions may differ. A path may also be compatible if two nodes are separated by a single anonymous node in one of the OCMs.

In the previous example, the two paths are compatible. The first component in both paths is B. This matches the destination exactly. The next component in the first path is C. This matches because it is separated from the B node only by a single anonymous node. If it was separated by multiple anonymous nodes or by another element, then the path is not compatible.

Example: OCM differs by additional element

Source OCM	Destination OCM
A {1,1} (,)	A {1,1} (,)
B {1,1} ()	B {1,1} ()
#text	#text
C {1,1} ()	C {1,1} ()
#text	#text
	D {1,1} ()
	#text

This operation performs these copies:

- A.B#text --> A.B.#text
- A.C#text --> A.C.#text

Note: If the additional elements in the destination exist before the source sequence is matched, then no copy is performed.

Example: OCM differs by additional element

Source OCM	Destination OCM
A {1,1} (,)	A {1,1} (,)
B {1,1} ()	X {1,1} ()
#text	#text
C {1,1} ()	B {1,1} ()
#text	#text
	C {1,1} ()
	#text

This operation fails.

Example: OCM differs by anonymous nodes

Source OCM	Destination OCM
A {1,1} (,)	A {1,1} (,)
B {1,1} ()	θ {1,-1} (,)
#text	B {1,1} ()
C {1,1} ()	#text
#text	C {1,1} ()
D {1,1} ()	#text
#text	D {1,1} ()
	#text

This operation performs these copies:

- A(|).B.#text --> A.θ(|).B.#text
- A(|).C.#text --> A.θ(|).C.#text
- A(|).D.#text --> A.θ(|).D.#text

The repetitions of the A element in the source are maintained in the repeating anonymous node θ in the destination.

Example: OCM differs by multiple anonymous nodes

Source OCM	Destination OCM
A {1,1} (,)	A {1,1} (,)
B {1,1} ()	θ {1,-1} (,)
#text	θ {0,-1} ()
C {1,1} ()	B {1,1} ()
#text	#text
D {1,1} ()	C {1,1} ()
#text	#text
	D {1,1} ()
	#text

This operation fails because there is more than one anonymous node separating A and B, C, and D.

Example: OCM differs by repetition

Source OCM	Destination OCM
A {1,-1} (,)	A {1,2} (,)
B {1,1} ()	B{1,1} ()
#text	#text
C {1,1} ()	C {1,1} ()
#text	#text
D {1,1} ()	D {1,1} ()
#text	#text

The source message can contain unlimited repetitions of A. The destination can contain only one or two repetitions of A. In this case, the first two repetitions of the source are copied to the destination. If the destination permits the same or more repetitions than the source, then all repetitions of the source are copied.

Example: OCM differs by element type (choice versus sequence)

Source OCM	Destination OCM
A {1,-1} (,)	A {1,-1} ()
B {1,1} ()	B{1,1} ()
#text	#text
C {1,1} ()	C {1,1} ()
#text	#text
D {1,1} ()	D {1,1} ()
#text	#text

The source message can contain unlimited repetitions.

This operation fails because A in the source is a sequence and A in the destination is a choice.

BULKCOPY

BULKCOPY can be considered a special case of PATHCOPY. It is accomplished by specifying the root node of the source and destination trees. This causes the entire message to be copied.

You can always use the BULKCOPY translation action. If this is used, then the engine interprets this as an invocation to PATHCOPY using the root as the paths for the source and destination.

UN/EDIFACT Configurator

UN/EDIFACT is the acronym for United Nations Electronic Data Interchange For Administration, Commerce, and Transport. This was created from a requirement for a single international EDI (Electronic Data Interchange) standard that was flexible enough for governments and private industries.

The UN/EDIFACT Configurator defines data elements, composite data structures, data segments, and messages for the UN/EDIFACT standard.

UN/EDIFACT files reside in the \$HCISITEDIR/formats/ directory.

Encoding separators

To enable you to set encoding separators for outbound messages, elements (an `ENCODE` keyed list) are added into the `SEPCHARS` keyed list. This makes the new `SEPCHARS` look similar to this example:

```
{FIELD "XXX"}
{COMPONENT "XXX"}
{ESCAPE "XXX"}
{SEGMENT "XXX"}
{DECIMAL "XXX"}
{ENCODE
  {
    {FIELD "XXX"}
    {COMPONENT "XXX"}
    {ESCAPE "XXX"}
    {SEGMENT "XXX"}
    {DECIMAL "XXX"}
  }
}
```

If you must change the segment separator of an outbound message, then set the `ENCODE.SEGMENT` value in an Xlate CALL action, as in this example:

```
set sepchars [xpmmetaget $xlateId SEPCHARS] keylset sepchars
ENCODE.SEGMENT "\r\n"
xpmmetaset $xlateId SEPCHARS $sepchars
```

You can also set the `ENCODE.FIELD`, `ENCODE.COMPONENT`, `ENCODE.ESCAPE`, and `ENCODE.DECIMAL` values in an Xlate CALL action to encode.

For example, this is for encoding a UN/EDIFACT message using `SEGMENT "$"` and `FIELD "#"` in a Tcl program:

```
set msgId [msgcreate]
msginsert $msgId "UNH+AA+AUTHOR'BGM++BB'DTM+CC'BUS+DD'"
set seps [msgmetaget $msgId SEPCHARS]
keylset seps ENCODE.FIELD "$"
keylset seps ENCODE.SEGMENT "#"
msgmetaset $msgId SEPCHARS $seps
set grmId [grmcreate -msg $msgId edifact 97B {} AUTHOR]
set msgId [grmencode $grmId]
```

UN/EDIFACT message formats

A UN/EDIFACT message is a series of sequential segments in the order that is specified in a message directory. It starts with the message header and ends with the message trailer. This is equivalent to a transaction set.

These segments consist of predefined functionally-related data elements or composite data elements whose values are identified by their sequential positions within the segment. Each segment starts with a segment tag and ends with a segment terminator. A segment can be, for example, a street or city name, that with other data elements define a precise concept, such as an address.

Data elements are the basic units of data.

A UN/EDIFACT message is defined as a single transaction set. A batch subsystem that parses and encodes deals with interchanges and groups.

A batch refers to a group of one or more messages. These are surrounded by header and trailer segments with no special characters required to delimit the individual messages inside the batch.

You can convert old translations and version variants by removing interchange segments. Most of the work is performed automatically by `hcirootcopy`, but translations that contain references to interchange control segments must be performed by hand.

To make the conversion, remove these segments that envelope a transaction set:

- UNA
- UNB
- UNG
- UNE
- UNZ

TPS procedures

These procedures are used to extract transaction sets from an interchange on the inbound side:

- `hciEDIFACTsplitinterchange`
- `hciEDIFACTbuildinterchange`

They then insert transaction sets into an interchange on the outbound side.

These procedures take a batch message and pull out the individual, encapsulated transaction sets in the process. Each encapsulated transaction set becomes a new message. The original batch message is removed. Configure these procedures in the Network Configurator.

Metadata field

The `SEPCHARS` metadata field key in the message object overrides default separator characters on a per message basis.

If the metadata field is `NULL`, then the defaults are used to parse or encode the message. If set, then it is a keyed list.

These are the expected keys:

- REPEAT
- ESCAPE
- DECIMAL

Note: EDIFACT refers to the `ESCAPE` key (`ESC`) as the release character.

The batch file defines the structure of a batch. It is found in this directory:

`%HCIROOT%\formats\standard\version directory`

In UN/EDIFACT, a batch is the same as an interchange. There is currently no support for defining batches inside a version variant.

Note: The finite state machine generator is written in C, instead of Tcl, making `.fsm` files unnecessary.

UN/EDIFACT data elements

A data segment is a logical grouping of data elements or composite data structures that are related to a specific type of data. For example, ADR Address, IDE Identity, PTY Priority. Each data segment consists of a segment identifier and one or more composite data structures or data elements.

The list box in the Defined Data Segments pane contains every segment defined in the selected variant, alphabetically sorted. Add or delete data segments in this pane.

Note: The standard UN/EDIFACT base definition cannot be deleted from within a variant.

For an existing data segment, click the segment name or specify a search string.

For a new data segment, click **New**.

- For **Data Segment Number**, specify a unique three-letter combination for the segment number.
- For **Data Segment Name**, specify a description for the segment.

Defining a new data element

This variant is only a copy of the standard UN/EDIFACT version. To create the new data element to add to this variant:

- 1 Click the **Data Elements** tab.
- 2 In the **Defined Data Elements** pane, click **New**.
- 3 Specify the **Data Element Number** and **Data Element Name**. Begin user-defined elements with 95 to 99 followed by two user-chosen numbers.
- 4 Click **OK**.
- 5 In the **Definition of the Selected Data Element** pane, select the **Type**, **Length**, and **Minimum Length** for the new data element.

UN/EDIFACT composite data structures

A data element represents a specific piece of data. A composite data structure consists of two or more data elements.

The list box in the **Defined Composite Data Structures** pane contains every composite defined in the selected variant, numerically sorted. Add or delete composite data structures in this pane.

Note: The standard UN/EDIFACT base definition cannot be deleted from within a variant.

For an existing composite selection, click the composite name or specify a search string.

For a new composite data structure, click **New**.

- For **Composite Data Structure Number**, specify a unique composite ID tag beginning with c or s followed by three numbers. Only composite data structure numbers can begin with a c or s.
- For **Composite Data Structure Name**, specify a descriptive name for the composite.

To change the composite data structure name, double-click the name in the list box. Specify a new name and click **OK**.

Defining the composite data structure

Use the Definition of the **Selected Composite Data Structure** pane to modify existing or define new composite data structures. Modify and define composite data structures by editing, deleting, or adding elements to the composite.

The list box lists the data elements included in the composite.

- 1 Click the composite data structure name in the **Defined Composite Data Structures** pane. The selected composite data structure's definition displays in the **Definition of the Selected Composite Data Structure** pane.
- 2 Designate where to place the new element by clicking the **Add/Paste New Elements** arrow and making a selection.
For example, to add an element after or before a particular element on the list, click the element where you require the insertion. Click **After Selected Element(s)** or **Before Selected Element(s)**.
- 3 Click the **Add** tool that is located below the Definition list. This opens the **Add Field to Composite** . Click **List** and double-click the element to add.
Click **OK** to place the element on the list at the location that was chosen under **Add/Paste New Elements**.
- 4 If necessary, then use the editing tools by clicking the appropriate tool that is located below the Definition of the Selected Composite Data Structure list box.
- 5 Specify if the element is required or conditional:
Mandatory specifies the element's inclusion is required.
Conditional specifies the element's inclusion is conditional.

UN/EDIFACT data segments

A data segment is a logical grouping of data elements or composite data structures that are related to a specific type of data. For example, ADR Address, IDE Identity, PTY Priority. Each data segment consists of a segment identifier and one or more composite data structures or data elements.

The list box in the **Defined Data Segments** pane contains every segment defined in the selected variant, alphabetically sorted. Add or delete data segments in this pane.

Note: The standard UN/EDIFACT base definition cannot be deleted from within a variant.

For an existing data segment selection, click the segment name and specify a search string.

For a new data segment, click **New**.

- For **Data Segment Number**, specify a unique three-letter combination for the segment number.
- For **Data Segment Name**, specify a description for the segment.

Defining the data segment

Use the **Definition of the Selected Data Segment** pane to modify existing or define new data segments. Modify a data segment definition by editing, deleting, or adding new elements to the segment.

The list box lists the data elements and composite data structures included in the data segment.

- 1 Click the segment name in the **Defined Data Segments** pane. The selected segment definition displays in the **Definition of the Selected Data Segment** pane.
- 2 Designate where to place the new data element or composite by clicking the **Add/Paste New Elements** arrow and making a selection.
For example, to add an element/composite after or before a particular element/composite, click the element where you require the insertion. Then, click **After Selected Element(s)** or **Before Selected Element(s)**.
- 3 Click the **Add** tool that is located below the Definition list. The **Add Data Element or Composite to Segment** dialog box is displayed.
- 4 Click **List** and double-click the data element or composite to add.
- 5 Click **OK** to place the element or composite on the list at the location that was chosen under **Add/Paste New Elements**.
- 6 If necessary, then use the editing tools by clicking the appropriate tool that is located below the Definition of the Selected Data Segment list box.
- 7 Specify if the segment's elements are required or conditional:
Mandatory specifies the element's inclusion is required.
Conditional specifies the element's inclusion is conditional.

UN/EDIFACT messages

The list box in the Defined Messages pane contains every message defined in the selected variant, alphabetically sorted.

Note: The standard UN/EDIFACT base definition cannot be deleted from within a variant.

For an existing message selection, click the segment name and specify a search string.

For a new message, click **New**.

- For **Message Name**, specify a unique three-letter combination for the segment number.
- For **Message Description**, specify a description for the segment.

Defining the message

Use the Definition of the **Selected Message** pane to modify existing or define new messages. Modify a message layout by editing, deleting, or adding new segments to the message definition.

When a new message is added, the UN/EDIFACT Configurator automatically inserts the required segments into the message.

- 1 Click the message name in the **Defined Messages** pane. The selected message's segments are displayed in the **Definition of the Selected Message** pane.
- 2 Designate where to place the new segment by clicking the **Add/Paste New Segments** arrow and making a selection.
For example, to add a segment or segments after or before a particular segment on the list, click the segment where you require the insertion. Then, click **After Selected Segment(s)** or **Before Selected Segment(s)**.
- 3 Click the **Add** tool that is located below the Definition list. The **Add Segment to Message** dialog box is displayed.
- 4 Click **List** and double-click the segment to add.
- 5 Click **OK** to place the segment on the list at the location that was chosen under **Add/Paste New Segments**.
- 6 If necessary, then use the editing tools by clicking the appropriate tool that is located below the Definition of the Selected Message pane:
 - For a single segment, click the segment to highlight it.
 - For a group of segments, click the first segment in the group and then shift-click the last segment. This selects the entire group.
- 7 Specify if the message's segments are optional or repeating:
 - **Optional** specifies the segment's inclusion is optional.
 - **Repeats** specifies whether the segment repeats.
If the segment is repeating, then select the **No Limit** check box or click the arrows to select the number of times the segment repeats.
 - **No Limit** specifies an unlimited number of repeats.

Configuring a UN/EDIFACT variant definition

This example shows how to define a UN/EDIFACT variant by creating a new variant containing a new segment and data element.

- 1 Select **File > New > version** in the UN/EDIFACT Configurator.
- 2 Specify an appropriate name.
- 3 Click **OK**.

Defining a new data element

This variant is only a copy of the standard UN/EDIFACT version. To create the new data element to add to this variant:

- 1 Click the **Data Elements** tab.
- 2 In the **Defined Data Elements** pane, click **New**.
- 3 Specify the **Data Element Number** and **Data Element Name**. Begin user-defined elements with 95 to 99 followed by two user-chosen numbers.

- 4 Click **OK**.
- 5 In the **Definition of the Selected Data Element** pane, select the **Type**, **Length**, and **Minimum Length** for the new data element.

Defining a new UN/EDIFACT data segment

- 1 On the **Data Segments** tab, in the Defined Data Segments pane, click **New**.
- 2 Specify the **Data Segment Number** and **Data Segment Name**.
- 3 Click **OK**.
- 4 Click **Add/Paste New Elements** to select the location to place the new element. Click the **Add** tool in the Definition of the Selected Data Segment pane to add the new data element to the segment.
- 5 Click **List** and double-click the new data element.
- 6 Click **OK**. The new segment or composite is added to the definition.
- 7 Repeat Steps 4 to 6 until all necessary elements are added.
- 8 Select the new segment in the Definition of the Selected Data Segment pane.
- 9 Select **Mandatory** or **Conditional** for each element.

Modifying the base UN/EDIFACT message layout

- 1 In the **Messages** tab, click the message in which to place the new segment in the Defined Messages pane. Its segments are displayed in the Definition of the Selected Message pane.
- 2 Click **Add/Paste New Segments** to select the location to place the new segment.
- 3 Click the **Add** tool in the Definition of the Selected Message pane.
- 4 Click **List** and double-click the new segment.
- 5 Click **OK**. The new segment is added to the definition.
- 6 In the Definition of the Selected Message pane, select **Optional** or **Repeats** to make the new segment an optional or repeatable item in the message layout.
To repeat, select the **No Limit** check box or specify the number of times in the field below the **No Limit** check box.
- 7 Save the file.

VRL Configurator

Variable-length Record Layout (VRL) Configurator defines how variable-length data is formatted within flat-record layouts.

VRL files reside in `$HCISITEDIR/formats/`.

VRL is similar to FRL in structure and purpose, but the underlying approach to accessing the data is different.

The main reason for VRL is the requirement to manipulate CSV (Comma Separated Values). VRL is used in a relational database environment that employs databases such as Oracle, Sybase, or DB2. Because the table and column definitions in these databases are usually variable in length, the data output from them must reflect this.

VRL supports quoted data that may contain field separators. For example, Smith, John D.

All separator characters for fields subfields and subsubfields that are found inside escape-sequence pairs are considered escaped. When retrieving data, they are not considered as separators.

Each VRL field definition can have a tag. If one field has a tag, then all fields must have tags. The Tag name can be the same as Name (for example, "last name" for both); or, it may be unique (for example, "last name" for Name and "ClientLN" for the Tag name).

Each VRL definition can contain optional subfields. Subfields are required in FRL. If a field contains subfields, then a subfield-separator character is required. It cannot be the same character as the field separator.

Each VRL subfield can contain subsubfields. If a subfield contains subsubfields, then a subsubfield separator character is required. It cannot be the same character as the field separator or the subfield separator.

The default value of **Max Width** (fields, subfields, and subsubfields) is "-1," which also disables **Max Width**. Using a value of "1" results in all data being truncated to one character long. For a specific width, specify the value. Incoming data larger than that value is truncated.

Tags

Tags permit a record definition where field position is not significant. Fields are accessed based on their tags.

In FRL, a given field has a fixed offset and length in the record to define where it begins and ends.

In normal VRL, a field has a fixed position in the record, where the separator characters define where one position ends and the next begins. Fields are sequential in nature.

In tagged VRL, a field has a tag and can happen anywhere in the record. Separators only serve to separate fields, not to define a position where the field is found.

Points to remember:

- Identical field names and tags are not required.
- Tagged fields cannot be addressed by position in the record.
- Tags are user-defined strings that must happen in the data for individual fields to be addressed.
- Tags must be unique within a record.
- Tags happen after the prefix string, but before the data.
- For input, a tag is not considered part of the data and is only used for parsing and addressing a field.
- For output, a field tag is not output if the field is not present.
- If the field is null (present, but zero length), then the field tag is output after the prefix string and before the postfix string.

Escape-sequence pair

Each VRL field definition can contain an optional escape-sequence pair. This defines a start escape-sequence character and an end escape-sequence character. Characters can be identical or different. This is useful when having field-separator characters in the data.

All separator characters (field/subfield/subsubfield separators) found inside escape-sequence pairs are considered escaped, and when retrieving data they are not considered as separators.

When encoding VRL data, if separator characters are found in the input data, they are escaped by embracing them with escape-sequence pair characters. For example, {Last, First} is encoded to {Last", " First}.

Note: The defined escape chars should not display in the message as normal content. If the chars have real meaning in the message, then you can define other chars as escape characters. Then, the engine can handle them as normal content.

VRL definition format

The format of a VRL definition file is similar to an FRL. It is composed of a limited set of directives and a list of keyed list entries. Each keyed list entry has a type that identifies its purpose.

Before configuring, specific transaction information is required, including:

- The types of transactions that are processed by each connection
- The record layouts for those transactions
- The types of data that are handled by each transaction
- The way that data is processed or translated

Then, use VRL Configurator to define:

- Variable-length flat-record layouts that format input and output data
- Separator characters to delimit positioned fields
- Tags to denote tagged fields
- Masks that define the layout of commonly used record fields
- Groups that define a group of related fields

Records can contain delimited fields, with user-defined separator characters or tagged fields, with user-assigned tags.

Global properties

These define the structure of the VRL message.

Global properties must be set before beginning VRL message configuration.

To configure user-defined global properties, select **Options > Global Properties** on the IDE's menu bar. This opens the **Global Properties** dialog box, where default field properties are shown.

This table shows the parameters in the **Global Properties** dialog box:

Parameter	Description
Field Separator	Identifies the user-defined field-separator character. The default is shown initially. Specify a different character, if necessary.

Parameter	Description
Default Subfield Separator	Shows the current default subfield separator. The default is shown initially. Specify a different character, if necessary.
Default SubSubfield Separator	Shows the current default subsubfield separator. The default is shown initially. Specify a different character, if necessary
Termination	Identifies the termination separator, which separates the different VRL segments from one another in a VRL string. This is because VRLs are usually used in HRLs as different segments. The default is shown initially. Specify a different character, if necessary. Termination characters are used when separating the repeated VRL messages in a HRL message. Therefore, a termination character must be used in a repeating VRL; otherwise, you cannot use that as a segment to iterate on.
Use Tagged Fields	Sets the option to identify fields with user-assigned tags. Tags are set in the Field, Mask, and Group Field Properties configuration panes.
Enable Outbound Escape Handling	Enables/disables the outbound escape handling, as the handling could have an effect on performance. Clear this if you are concerned with the performance of the outbound VRL encoding. This feature is cleared by default. If your message uses tags, then select the Use Tagged Fields option on the Global Properties dialog box before beginning each new layout.

Configuring the VRL field

Variable-length flat-records consist of one or more fields, which can have one or more subfields (optional), which can have one or more subsubfields.

A VRL must match the data to or from the ancillary system, or incorrect results are produced.

- 1 To create a new file or open an existing one, select **Define > Define Fields**.
To create a new field, click the **New Field** tool. This places a new field folder in the Layout pane.
To reconfigure an existing field, open the file to edit. In the Layout pane, click the field to reconfigure. Its current properties display in the Field Properties pane.
- 2 Specify the field name.

- 3 For **Prefix**, specify the characters that precede this field. This separates it from the previous field in the record layout. This value is optional.
- 4 For **Postfix**, specify the characters that follow this field. This separates it from the next field in the record layout. This value is optional.
- 5 For **Min Width**, specify the minimum width of the field, not including any prefix.
- 6 For **Max Width**, specify the maximum width of the field, not including any prefix. To disable, specify a value of -1. This is the default. For a specific width, specify the value. Incoming data larger than that value are truncated.
- 7 For **SF Sep**, specify a subfield-separator character.
- 8 For **Escape Pair**, specify a start escape-sequence character and an end escape-sequence character. Escape-sequence characters are useful when using field-separator characters in the data. The default is "". Unique or identical characters are permitted.
 All separator characters that are found inside escape-sequence pairs are considered escaped. When retrieving data, they are not considered as separators.
 When encoding VRL data, if separator characters are found in the input data, they are escaped by embracing them with escape-sequence pair characters. For example, {Last, First} is encoded to {Last", "First}.
 For **Tag**, specify a user-defined string. If one field has a tag, then all fields must have tags. The **Tag** name can be the same as **Name** (for example, "last name" for both. Or, it can be unique. For example, "last name" for Name and "ClientLN" for the Tag name.
 This option is available only when **Use Tagged Fields** is selected on the **Global Properties** dialog box. See [Global properties](#).
- 9 Select the validation:
Validate at Fetch validates the field whenever this record layout is retrieved or used for an input or output operation. With this selected, messages with an invalid field are parsed, but a warning is given.
Validate at Parse validates the field when the data is parsed. This option forces validation even when the field is not used. With this selected, messages with an invalid field fail when parsed.
- 10 Select how to validate the subfield data whenever this record layout is used as an input or output record.
Existence validates the existence of the data, but not its content.
Contents validates the type of data, but not its existence.
Existence and **Contents** validate both the existence and the content of the data.
- 11 Select how to define the field:
Normal defines the field format by the current field entries, instead of a mask or group. This is the default setting.
Mask defines the current field with a mask.
Group defines the current field with a group.

Configuring the VRL subfield properties

- 1 To create a new subfield, select the field in which to place the subfield and click **New Subfield**.
- 2 To reconfigure an existing subfield, open the field folder and select the subfield to reconfigure.

- 3 For **Data Type**, click the arrow to select from a menu of data types.
- 4 For **Prefix**, specify the characters that precede this subfield. This separates it from the previous subfield in the record layout. This value is optional.
- 5 For **Postfix**, specify the characters that follow this subfield. This separates it from the next subfield in the record layout. This value is optional.
- 6 For **Min Width**, specify the minimum width of the subfield, not including any prefix.
- 7 For **Max Width**, specify the maximum width of the subfield, not including any prefix. To disable, specify a value of -1. This is the default. For a specific width, specify the value. Incoming data larger than that value is truncated.
- 8 For **SSF Sep**, specify a subsubfield-separator character.
This must be different from the global **Default Subfield Separator**, which is listed on the **Global Properties** dialog box. This is found at **Options > Global Properties**.
This must also be different from the field **SF Sep**, subfield separator, character.
- 9 Select how to validate the subfield data whenever this record layout is used as an input or output record.
Existence validates the existence of the data, but not its content.
Contents validates the type of data, but not its existence.
Existence and **Contents** validate both the existence and the content of the data.

Configuring the VRL subsubfield properties

Each VRL subfield can contain subsubfields. If a subfield contains subsubfields, then a subsubfield-separator character is required. This is defined in Subfield Properties definition. Do not use the same subsubfield-separator character as the field separator or the subfield separator.

- 1 To create a new subsubfield, select the subfield in which to place the subsubfield and click **New SubSubfield**.
To reconfigure an existing subsubfield, open the field and subfield containing the subsubfield. Edit as necessary.
- 2 For **Data Type**, click the arrow to select from a menu of data types.
- 3 For **Prefix**, specify the characters that precede this subsubfield. This separates it from the previous subsubfield in the record layout. This value is optional.
- 4 For **Postfix**, specify the characters that follow this subsubfield. This separates it from the next subsubfield in the record layout. This value is optional.
- 5 For **Min Width**, specify the minimum width of the subsubfield, not including any prefix.
- 6 For **Max Width**, specify the maximum width of the subsubfield, not including any prefix. To disable, specify a value of -1. This is the default. For a specific width, specify the value. Incoming data larger than that value is truncated.
- 7 For validation, select how to validate the subsubfield data whenever this record layout is used as an input or output record.
Existence validates the existence of the data, but not its content.
Contents validates the type of data, but not its existence.
Existence and **Contents** validate both the existence and the content of the data.

Configuring the VRL mask

In this way, masks can be created to define field formats used in multiple places. Each mask is composed of one or more subfields, which contain the actual data within the record. A mask is a template that is used to define data fields that have the same layout. The mask is comparable to a template field definition that is used repeatedly.

For example, a client name mask defines a template of what a client name field looks similar to (that is, its subfield layout).

Defining a mask is similar to defining a field within a record layout.

- 1 To create a new file or open an existing one, select **Define > Define Masks**.
- 2 Select **New Field** from the toolbar. This places a new mask in the Layout pane.
- 3 For **Name**, specify the mask name.
- 4 For **Data Type** A mask is a template that is used to define data fields that have the same, click the arrow to select from a menu of data types.
- 5 For **Prefix**, specify the characters that precede this field, to separate it from the previous field in the record layout. This value is optional.
- 6 For **Postfix**, specify the characters that follow this field, to separate it from the next field in the record layout. This value is optional.
- 7 For **Min Width**, specify the minimum width of the field, not including any prefix.
- 8 For **Max Width**, specify the maximum width of the field, not including any prefix. To disable, specify a value of -1 (default). For a specific width, specify the value. Incoming data larger than that value is truncated.
- 9 For **SF Sep**, specify a subfield-separator character.
- 10 For **Escape Pair**, specify a start escape-sequence character and an end escape-sequence character. Escape-sequence characters are useful when using field-separator characters in the data. The default is "". Unique or identical characters are permitted.

All separator characters that are found inside escape-sequence pairs are considered escaped, and when retrieving data they are not considered as separators.

When encoding VRL data, if separator characters are found in the input data, they are escaped by embracing them with escape-sequence pair characters. For example, {Last, First} is encoded to {Last", "First}.

- 11 For **Tag**, specify a user-defined string. If one field has a tag, then all fields must have tags. The **Tag** name can be the same as **Name**. For example, "last name" for both. Or, it may be unique. For example, "last name" for **Name** and "ClientLN" for the **Tag** name.

This option is available only when **Use Tagged Fields** is selected on the Global Properties dialog box. See [Global properties](#).

- 12 Select the validation.

Validate at Fetch validates the field whenever this record layout is retrieved or used for an input or output operation. With this selected, messages with an invalid field are parsed, but a warning is given.

Validate at Parse validates the field when the data is parsed. This option forces validation even when the field is not used. With this selected, messages with an invalid field fail when parsed.

- 13 Select how to validate the subsubfield data whenever this record layout is used as an input or output record.

Existence validates the existence of the data, but not its content.

Contents validates the type of data, but not its existence.

Existence and **Contents** validate both the existence and the content of the data.

Configuring the VRL mask subfield properties

VRL mask definitions can contain optional subfields. If a field contains subfields, then a unique subfield-separator character is required. This is defined in Mask Properties definition.

- 1 To create a new mask subfield, select the mask folder in which to place the subfield and click **New Subfield**.
To reconfiguring an existing mask subfield, select the subfield to reconfigure and edit as necessary.
- 2 For **Data Type**, click the arrow to select from a menu of data types.
- 3 For **Prefix**, specify the characters that precede this subfield. This separates it from the previous subfield in the record layout. This value is optional.
- 4 For **Postfix**, specify the characters that follow this subfield. This separates it from the next subfield in the record layout. This value is optional.
- 5 For **Min Width**, specify the minimum width of the subfield, not including any prefix.
- 6 For **Max Width**, specify the maximum width of the subfield, not including any prefix. To disable, specify a value of -1. This is the default. For a specific width, specify the value. Incoming data larger than that value is truncated.
- 7 For **SSF Sep**, specify a subsubfield-separator character
This must be different from the global **Default Subfield Separator**, which is listed on the Global Properties dialog box. This is located at **Options > Global Properties**.
See [Global properties](#).
This must also be different from the field **SF Sep** character.
- 8 For validation, select how to validate the subfield data whenever this record layout is used as an input or output record.
Existence validates the existence of the data, but not its content.
Contents validates the type of data, but not its existence.
Existence and **Contents** validate both the existence and the content of the data.

Configuring the VRL mask subsubfield properties

Each VRL subfield can contain subsubfields. If a subfield contains subsubfields, then a subsubfield-separator character is required. This is defined in Subfield Properties definition. Do not use the same subsubfield-separator character as the field separator or the subfield separator.

- 1 To create a new mask subsubfield, select the subfield in which to place the subsubfield and click **New Subsubfield**.

To reconfigure an existing mask subsubfield, select the subsubfield to be reconfigured and edit as necessary.

- 2 For **Data Type**, click the arrow to select from a menu of data types.
- 3 For **Prefix**, specify the characters that precede this subsubfield. This separates it from the previous subsubfield in the record layout. This value is optional.
- 4 For **Postfix**, specify the characters that follow this subsubfield. This separates it from the next subsubfield in the record layout. This value is optional.
- 5 For **Min Width**, specify the minimum width of the subsubfield, not including any prefix.
- 6 For **Max Width**, specify the maximum width of the subsubfield, not including any prefix. To disable, specify a value of -1. This is the default. For a specific width, specify the value. Incoming data larger than that value is truncated.
- 7 For validation, select how to validate the subfield data whenever this record layout is used as an input or output record:

Existence validates the existence of the data, but not its content.

Contents validates the type of data, but not its existence.

Existence and **Contents** validate both the existence and the content of the data.

Configuring the VRL group field properties

Both groups and masks serve as templates for defining the data fields in a record layout. A mask defines only a single data field. A group defines many related fields.

A group is a set of logically related data fields and is used to define a multiple field format in multiple places. For example, an address group contains fields for street, city, state, and zip.

After defining a group, you can use it as a template for declaring groups of data fields in the record layout.

- 1 Create a new group by selecting **Define > Define Groups** and clicking **New Group**.
- 2 After giving the group a name, click **New Field**.
- 3 For **Name**, specify the group field name.
- 4 For **Data Type**, click the arrow to select from a menu of data types.
- 5 For **Prefix**, specify the characters that precede this field. This separates it from the previous field in the record layout. This value is optional.
- 6 For **Postfix**, specify the characters that follow this field. This separates it from the next field in the record layout. This value is optional.
- 7 For **Min Width**, specify the minimum width of the field, not including any prefix.
- 8 For **Max Width**, specify the maximum width of the field, not including any prefix. To disable, specify a value of -1. This is the default. For a specific width, specify the value. Incoming data larger than that value is truncated.
- 9 For **SF Sep**, specify a subfield separator character.
- 10 For **Escape Pair**, specify a start escape-sequence character and an end escape-sequence character. Escape-sequence characters are useful when using field-separator characters in the data. The default is "". Unique or identical characters are permitted.

All separator characters found inside escape-sequence pairs are considered escaped, and when retrieving data they are not considered as separators.

When encoding VRL data, if separator characters are found in the input data, they are escaped by embracing them with escape-sequence pair characters. For example, {Last, First} is encoded to {Last", "First}.

- 11** For **Tag**, specify a user-defined string. If one field has a tag, then all fields must have tags. The Tag name can be the same as Name. For example, "last name" for both. Or, it may be unique. For example, "last name" for Name and "ClientLN" for the Tag name.

This option is available only when **Use Tagged Fields** is selected on the **Global Properties** dialog box. See [Global properties](#).

- 12** For validation, select whether fields are validated at parse time or retrieve time.

For **Validate at Fetch**, click to validate the field whenever this record layout is retrieved or used for an input or output operation. With this selected, messages with an invalid field are parsed, but a warning is given.

For **Validate at Parse**, click to validate the field when the data is parsed. This option forces validation even when the field is not used. With this selected, messages with an invalid field fail when parsed.

- 13** Select how to validate the subfield data whenever this record layout is used as an input or output record.

Existence validates the existence of the data, but not its content.

Contents validates the type of data, but not its existence.

Existence and **Contents** validate both the existence and the content of the data.

- 14** Select how to define the field.

For **Normal**, click to define the field format by the current field entries, instead of a mask or group. This is the default setting.

For **Mask**, click to define the current field with a mask. Click the arrow to open a menu listing available masks.

Configuring the VRL group subfield properties

Each field of the group is composed of one or more subfields. Subfields contain the actual data within the record. The subfields of a field are the same as the subfields of a record-layout data field.

- 1** To create a new group subfield, select the field folder in which to place the new subfield and click **New Subfield**.
- 2** For **Data Type**, click the arrow to select from a menu of data types.
- 3** For **Prefix**, specify the characters that precede this subfield. This separates it from the previous subfield in the record layout. This value is optional.
- 4** For **Postfix**, specify the characters that follow this subfield. This separates it from the next subfield in the record layout. This value is optional.
- 5** For **Min Width**, specify the minimum width of the subfield, not including any prefix.
- 6** For **Max Width**, specify the maximum width of the subfield, not including any prefix. To disable, specify a value of -1. This is the default. For a specific width, specify the value. Incoming data larger than that value is truncated.

- 7 For **SSF Sep**, specify a subsubfield-separator character.
- 8 For validation, select how to validate the subfield data whenever this record layout is used as an input or output record.
Existence validates the existence of the data, but not its content.
Contents validates the type of data, but not its existence.
Existence and **Contents** validate both the existence and the content of the data.

Configuring the VRL group subsubfield properties

Each VRL subfield can contain subsubfields. If a subfield contains subsubfields, then a subsubfield-separator character is required. This is defined in Subfield Properties definition. Do not use the same subsubfield-separator character as the field separator or the subfield separator.

- 1 To create a new group subsubfield, select the subfield in which to place the subsubfield and click **New Subsubfield**.
- 2 For **Data Type**, click the arrow to select from a menu of data types.
- 3 For **Prefix**, specify the characters that precede this subsubfield. This separates it from the previous subsubfield in the record layout. This value is optional.
- 4 For **Postfix**, specify the characters that follow this subsubfield. This separates it from the next subsubfield in the record layout. This value is optional.
- 5 For **Min Width**, specify the minimum width of the subsubfield, not including any prefix.
- 6 For **Max Width**, specify the maximum width of the subsubfield, not including any prefix. To disable, specify a value of -1. This is the default. For a specific width, specify the value. Incoming data larger than that value is truncated.
- 7 For validation, select how to validate the subsubfield data whenever this record layout is used as an input or output record:
Existence validates the existence of the data, but not its content.
Contents validates the type of data, but not its existence.
Existence and **Contents** validate both the existence and the content of the data.

VRL record layout definition example

Before configuring, specific transaction information is required, including:

- The types of transactions that are processed by each connection
- The record layouts for those transactions
- The types of data that are handled by each transaction
- The way that data is processed or translated

Note: Records can contain delimited fields with user-defined separator characters, or tagged fields with user-assigned tags.

Access `hcivrltest` directly from the command line or through the Testing Tool.

Use the Testing Tool's **VRL** tab to test variable-record layout configurations. This test reads, parses, and displays messages, ensuring that configurations match the input data.

Modifying the global properties

Global properties define the structure of the VRL message.

Note: Global properties must be set before beginning a VRL message configuration. If your message uses tags, then select the **Use Tagged Fields** option on the **Global Properties** dialog box before beginning each new layout.

- 1 Open the VRL Configurator and select **Options > Global Properties**. Configure the values as required.
- 2 When all values are set, click **Apply**.

Creating a variable-length flat record layout

Note: The fields and subfields must be in precise order in the layout pane for the VRL layout to parse or encode data properly. Or, you can select **Use Tagged Fields** in the **Global Properties** dialog box and specify a tag for each field and subfield.

- 1 Click **New Field** on the VRL Configurator toolbar. A new field displays in the Layout pane.
- 2 In the Field Properties area, specify the name of the field in **Name**.
- 3 Select the type of data that goes in the field from the **Data Type** menu.
- 4 Click **Apply**. The name of the selected field changes in the Layout pane.
- 5 Select the appropriate subfield beneath the field in the Layout pane. The Subfield Properties area displays.
- 6 Update the Subfield Properties as necessary and click **Apply**.
- 7 Repeat steps 1-6 to create the VRL layout with the necessary field and subfield properties.
- 8 Save the layout.

Testing the layout

- 1 Click **Test** on the toolbar. The Testing Tool opens with the VRL tab active.
- 2 Select the VRL layout file from the **VRL File**.
- 3 In the Test Options area, click **Open** file next to **Choose Data File**.
- 4 Navigate to the data folder and select the data file with which to test. Click **Open**. The path to the data file displays in **Choose Data File**.
- 5 Select the **Detail Level** from the list.
- 6 Select **process one record**.
- 7 In the Preview Command to Issue area, click **Run Command**. The output displays in the Result area.

WSDL2XSD tool

The XSD WSDL tool is integrated into the IDE as a wizard. The wizard guides you in generating and compiling the XSD file, and generating the WSDL file from the XSD file.

The wizard can also generate the WS-Policy by consuming an existing WSDL, if it is provided.

For information, see **Web Services Consumer Wizard > XSD WSDL tool** in the *Cloverleaf Application Adaptor-Web Services User Guide*.

X12 Configurator

An X12 message is defined as a single transaction set. It is a batch subsystem that parses and encodes, dealing with interchanges and groups.

A batch refers to a group of one or more messages. These messages are surrounded by header and trailer segments with no special characters required to delimit the individual messages inside the batch.

New segments can be optionally added at the top or at the end of the message.

Transaction structure

A transaction set is composed of:

- A transaction set header control segment
- One or more data segments
- A transaction set trailer control segment

The transaction set might also include a transaction security header control segment and transaction security trailer control segment. These are for sending the data within the transaction envelope in an authenticated or encrypted form.

Each segment is composed of a unique segment ID, one or more logically-related data elements or composite data structures, and a segment terminator.

A data element in the transaction set header identifies the type of transaction set.

Composite data structures are composed of one or more logically-related component data elements each, except the last, followed by a component element separator.

The data segment directory entry referenced by the data segment identifier defines the sequence of data elements and composite data structures in the segment. It also defines any interdependencies that might exist.

The composite data structure directory entry referenced by the composite data structure number defines the sequence of component data elements in the composite data structure.

A functional group contains one or more related transaction sets that are preceded by a functional group header control segment. This is terminated by a functional group trailer control segment. The functional group might also include a functional security header control segment and functional security trailer control

segment. These can be used to send the data within the functional envelope in an authenticated or encrypted form, or both.

Data elements, composite data structures, data segments, and transaction sets can be copied and pasted across variants. When one of these is highlighted, right-clicking opens a menu with **Copy**, **Paste**, and **Delete** options. Multiple selections are permitted.

Note: The **Paste** action does not apply between different HMD formats.

Syntax notation

The syntax for X12 is defined in a BNF (Backus-Naur Form). Each definition is accompanied by explanatory text. Sometimes, actual use, that is, the actual data stream, of an X12 structure might not be the same as the definition. This is because some structures are optional.

All data element values, except those of the binary data element, are constructed using a character set. The character set of this standard is grouped according to common characteristics.

The X12 standards are graphic-character oriented, so any common character-encoding schemes may be used as long as a common mapping is available. No collating sequence is to be assumed in any definition used in the standard. This is because no single character code is specified and no other means of specifying a sequence is provided.

Delimiters consist of two levels of separators and a terminator. The delimiters are an integral part of the transferred data stream. Delimiters are specified in the interchange header. They are not used in a data element value elsewhere in the interchange, with the exception of their use in the binary data element. From highest to lowest level, the separators and terminator are:

- Segment terminator
 <tr> ::
- Data element separator
 <gs> ::
- Component element separator
 <us> ::

To set encoding separators for outbound messages, elements and an `ENCODE` keyed list are added to the `SEPCHARS` keyed list. This make the new `SEPCHARS` look similar to:

```
{FIELD "XXX"}
{COMPONENT "XXX"}
{SEGMENT "XXX"}
{REPEAT "XXX"}
{ENCODE
  {
    {FIELD "XXX"}
    {COMPONENT "XXX"}
    {SEGMENT "XXX"}
    {REPEAT "XXX"}
  }
}
```

To change the segment separator of an outbound message, set the `ENCODE.SEGMENT` value in an `Xlate CALL` action:

```
set sepchars [xpmmetaget $xlateId SEPCHARS] keylset sepchars
ENCODE.SEGMENT "\r\n"
xpmmetaset $xlateId SEPCHARS $sepchars
```

You can also set the `ENCODE.FIELD`, `ENCODE.COMPONENT`, and `ENCODE.REPEAT` values in an `Xlate CALL` action to encode.

This is an example for encoding an X12 message using `SEGMENT "@"` and `FIELD "%"`. This is only in a Tcl program.

```
set msgId [msgcreate]
msginsert $msgId "ST*835~BPR*RA*9491.85~"
set seps [msgmetaget $msgId SEPCHARS]
keylset seps ENCODE.SEGMENT "@"
keylset seps ENCODE.FIELD "%"
msgmetaset $msgId SEPCHARS $seps
set grmId [grmcreate -msg $msgId x12 004030 {} 835]
set msgId [grmencode $grmId]
```

Data elements

The data element is the smallest named unit of information in the standard. Data elements are identified as "simple" or "component".

A data element that happens as an ordinaly-positioned member of a composite data structure is identified as a component data element.

A data element that happens in a segment outside of the defined boundaries of a composite data structure is identified as a simple data element.

The distinction between simple and component data elements is only a matter of context because a data element can be used in both capacities.

These definitions apply equally to each class of data element:

```
simple_data_element :: data_element
component_data_element :: data_element
data_element :: numeric | decimal_number | id |
string | date | time | binary |
fixed_length_string
```

Composite data structure

The composite data structure is an intermediate unit of information in a segment. This consists of two or more component data elements preceded by a data element separator. In use a composite data structure may display as only one component data element. Each component data element with the composite data structure, except the last, is followed by a component element separator. The final component data element is followed by the next data element separator or the segment terminator. Trailing component data element separators `<us>` are suppressed.

Composite data structures are defined in a composite data structure directory. The directory defines each composite data structure by its name, purpose, reference identifier, and included component data elements in a specified sequence.

A composite data structure is constructed in this manner:

Definition:

```
composite_data_structure ::= component_data_element
<us> component_data_element
{<us> component_data_element}
```

Usage:

```
composite_data_structure ::= {[component_data_element]
<us>} component_data_element
```

Data segment

The data segment is an intermediate unit of information in a transaction set. A data segment consists of a segment identifier; one or more composite data structures or simple data elements, each preceded by a data element separator; and ending with a segment terminator. Trailing data element separators <gs> are suppressed.

Data segments are defined in a data segment directory. The directory defines each segment including the segment's name, purpose, and identifier. The directory also defines composite data structures and data elements that a segment contains in their specified order.

A data segment is constructed in this manner:

Definition:

```
data_segment ::= seg_id <gs> data_segment_unit
{<gs> data_segment_unit} <tr>
```

Usage:

```
data_segment ::= seg_id {<gs> [data_segment_unit]}
<gs> data_segment_unit <tr>
data_segment_unit ::= simple_data_element |
composite_data_structure
```

Transaction set

A transaction set is a semantically meaningful unit of information exchanged between trading partners.

The transaction set consists of:

- A transaction set header segment.
- A transaction security header segment. This is optional.
- One or more data segments and loop-control segments, if bounded loops exist, in a specified order.
- A transaction security trailer segment whenever the security header segment is present.
- A transaction set trailer segment.

A transaction set is constructed in this manner:

```
transaction_set ::= unsecured_trans_set |
secured_trans_set
unsecured_trans_set ::= trans_set_header
```

```
data_segment_group {data_segment_group}
trans_set_trailer
secured_trans_set ::= trans_set_header
trans_security_header data_segment_group
{data_segment_group} trans_security_trailer
trans_set_trailer
```

HIPAA

HIPAA-004010 is an X12 version for health care transaction exchanges that follows national guidelines for EDI (Electronic Data Interchange) established for this purpose.

HIPAA is the acronym for Health Insurance Portability and Accountability Act. This act was created to improve productivity of the American health care system. It encourages development of information systems that are based on the exchange of standard management and financial data using EDI.

The HIPAA implementation guidelines introduce a new concept to the X12 standard. That is, there are two different transaction sets that are called 278 and three different transaction sets that are called 837. This creates a challenge to the system to parse these messages correctly. Two or more message types cannot be used with the same identifier. They cannot be defined in one version of a message format or its variant. In other words, there cannot be two 278 messages defined in HIPAA-004010.

To solve this problem, there is a pre-defined HIPAA variant concept.

The HIPAA-004010 message format has messages for:

- 270: Eligibility, Coverage, or Benefit Inquiry
- 271: Eligibility, Coverage, or Benefit Information
- 276: Health Care Claim Status Request
- 277: Health Care Claim Status Notification
- 820: Payment Order/Remittance Advice
- 834: Benefit Enrollment and Maintenance
- 835: Health Care Claim Payment/Advice

Pre-defined variants

The first pre-defined variant called HIPAA-004010 v1 has additional messages for:

- 278: Health Care Services Review: Request for Review
- 837: Health Care Claim: Professional

The second pre-defined variant called HIPAA-004010 v2 has additional messages for:

- 278: Health Care Services Review: Response to Request for Review
- 837: Health Care Claim: Dental

The third pre-defined variant called HIPAA-004010 v3 has an additional message for 837: Health Care Claim: Institutional

When a new site is initialized, there are three X12 HIPAA-004010 variants created automatically. The division of these variants to contain different message types is arbitrary. Implementers can rearrange different message types to create new variants based on their application requirements.

Configuring an X12 data element

A data element represents a specific piece of data, such as a billing code or reference number, and has a specific set of attributes.

Add or modify data elements within specific variants. Select an existing variant or create a new one.

The list box in the Defined Data Elements pane contains every element defined in the selected variant. Add or delete data elements in this pane.

Note: The standard X12 base definition for a data element cannot be deleted from within a variant.

- 1 To create a new data element, click **New**.
- 2 For **Data Element Number**, specify a number that is unique among all elements in the current variant. User-defined elements should begin with a Z followed by a user-chosen number.
- 3 For **Data Element Name**, specify a descriptive name for the element.
- 4 Click the **Type** arrow to select from a list of types.
- 5 Click the **Length** buttons to select the maximum number of characters that one instance of the data element can occupy in a message. Include component separator characters in this calculation. Select **Unlimited** for an unlimited field length.
- 6 Click the **Minimum Length** arrow buttons to select the minimum length of the added or edited field.

Configuring the X12 composite data structure

A composite data structure consists of two or more component data elements, preceded by a data element separator, with a specific set of attributes.

Add or modify data elements within specific variants. Select an existing variant or create a new one.

The list box in the Defined Composite Data Structures pane contains every composite defined in the selected variant.

Use the Definition of the Selected Composite Data Structure pane to modify existing or define new composites. Modify and define composites by editing, deleting, or adding elements to the composite.

The list box lists the data elements included in the composite.

Note: The standard X12 base definition for a composite cannot be deleted from within a variant.

- 1 To create a new composite data structure, click **New**. This opens the **Create New Composite Data Structure** dialog box.
- 2 For **Composite Data Structure Number**, specify a number that is unique among all composites in the current variant. The X12 standard requires composite data structure numbers to begin with an S or C.
- 3 For **Composite Data Structure Name**, specify a descriptive name for the composite.
- 4 For existing composite data structures, designate where to place the new element by clicking the **Add/Paste New Elements** arrow and making a selection.
To add an element after or before a particular element on the list, first click the element where the insertion is required. Then, click **After Selected Element(s)** or **Before Selected Element(s)**.
- 5 In the definition panel, click **Add**. This opens the **Add Field to Composite** dialog box.

Click **List** and double-click the field to add.

Click **Apply** and then **OK**.

- 6 If necessary, then use the editing tools to refine the definition.
- 7 Specify whether the new element is **Mandatory**, **Optional**, or **Conditional**.

Configuring the X12 data segment

A data segment is a logical grouping of data elements or composite data structures.

Add or modify data elements within specific variants. Select an existing variant or create a new one.

Use the Definition of the Selected Data Segment pane to modify existing or define new segments. Modify and define segments by editing, deleting, or adding new elements to the segment.

- 1 To create a new segment click **New**. This opens the **Create New Data Segment** dialog box.
- 2 For **Data Segment Number**, specify a unique two- or three-letter/number combination for the segment name.
- 3 For **Data Segment Name**, specify a description for the segment.
- 4 For existing data segments, designate where to place the element or composite by clicking the **Add/Paste New Elements** arrow and making a selection.
For example, to add an element or composite after or before a particular element on the list, click the element where the insertion is required. Then, click **After Selected Element(s)** or **Before Selected Element(s)**.
- 5 In the Definition panel, click **Add**. This opens the **Add Data Element or Composite to Segment** dialog box.
- 6 Click **List** and double-click the data element or composite to add.
- 7 Click **Apply** and then **OK**.
- 8 If necessary, then use the editing tools to refine the definition.
- 9 Specify whether the new element is **Mandatory**, **Optional**, or **Conditional**.

Configuring the X12 transaction set

A transaction set consists of any number of segments. Each transaction set requires a ST segment at the beginning and a SE segment at the end. This is provided automatically.

Add or modify data elements within specific variants. Either select an existing variant or create a new one.

Use the Definition of the Selected Transaction Set pane to modify existing or define new transaction sets. Modify a transaction set by editing, deleting, or adding new segments to the transaction set definition.

When a new transaction set is added, the X12 Configurator automatically inserts the required segments into the transaction set. The segments added are placed between the first ST and last SE segments.

- 1 To create a new transaction set, click **New**. This opens the **Create New Transaction Set** dialog box.

- 2 For **Transaction Set Name**, specify a unique number for the transaction set type.
- 3 For **Transaction Set Description**, specify a descriptive name for the transaction set.
- 4 For existing transaction sets, designate where to place the segments by clicking the **Add/Paste New Segments** arrow and making a selection.
For example, to add a segment or segments after or before a particular segment on the list, click the segment where the insertion is required. Then, select **After Selected Segment(s)** or **Before Selected Segment(s)**.
- 5 In the Definition panel, click **Add**. This opens the **Add Segment to Transaction Set** dialog box.
- 6 Click **List** and double-click the segment to add.
- 7 Click **Apply** and then **OK**.
- 8 Specify if the transaction set's segments are optional or repeating.

Creating optional and repeating groups

For X12 transaction sets, select the **Optional** or **Repeats** check boxes to set the optional and repeat attributes of the transaction set's segments.

You can do this for a single segment or a segment group. For a segment group, shift-click to select a group of segments from the current message definition. Then, select the **Optional** or **Repeats** check boxes to make the entire selection optional or repeating. Specify the number of repeats, if required.

Select the **No Limit** check box to specify an unlimited number of repeats. When **No Limit** is cleared, you can select the number of times the segment repeats.

This table shows the symbols used in the Definition of the Selected Transaction Set panel:

Symbols	Description
[]	Optional segments are enclosed in brackets.
{ }	Repeating segments are enclosed in braces.
{ [] }	Repeating optional segments are enclosed in braces and brackets.
[{ }]	Optional repeating segments are enclosed in brackets and braces.

If "[" or "{" is clicked, then everything up to the closing "]" or "}" is selected.

If an unbalanced group is selected, that is, only part of an optional or repeating group, then the **Cut**, **Copy**, **Optional**, and **Repeats** functions are unavailable.

XML Package Manager

XML, eXtensible Markup Language, is a general purpose markup language that is widely used across the world to share structured data across different information systems. Support is provided for XML as a built-in message format.

To use this feature , you must:

- Define XML definition files for the expected data.
See [XML definition files](#).
- Deploy these files into packages in the system.
See [File management](#).
- Compile these files for the system built-in processing.
See [Compiling/Recompiling](#).

The XML Package Manager facilitates deployment and compiling. It does this by providing a means through which DTD/Schema/DTD-containing-XML files are managed and compiled for built-in processing by the system.

The Xerces DOM parser is used in Cloverleaf. This requires that, in a DTD file, an element can be declared only once. Previous versions of Xerces DOM parser do not have this restriction.

For example, this DTD is not valid, because element a is declared twice:

```
<!ELEMENT foo (A,B)>
<!ELEMENT A (a,b)>
<!ELEMENT a (#PCDATA)>
<!ELEMENT b (#PCDATA)>
<!ELEMENT B (a,c)>
<!ELEMENT a (#PCDATA)>
<!ELEMENT c (#PCDATA)>
```

Points to remember

- OCM files are not shown in the server tree view at the package level.
- The deletion of a parent root DTD/Schema/DTD-containing-XML file automatically deletes the associated OCM file.
- The completion of a Cut of a parent root DTD/Schema/DTD-containing-XML file automatically deletes the associated OCM file.
- OCM files that have somehow been 'orphaned' from their parent root DTD/Schema/DTD-containing-XML file at the package level are automatically removed.
- OCM creation can be performed on multiple files at one time. A "busy" icon is shown until all indicated OCM creations are satisfied by success or failure of the process.

The XML Package Manager generally prevents the inadvertent copy of the hidden OCM files from one location to another.

XML terms

This table shows the terms used in working with **XML Package Manager**:

Term	Description
XML	<p>This is an acronym for eXtensible Markup Language. This is a protocol (meta-language) for marking the structure of an arbitrary set of data.</p> <p>This data can be anything that contains an internal organization of parts that contribute to the meaning of the whole. For example, documents, data streams (messages), and computer programs.</p> <p>The intent of XML is to delineate a set of simplified rules for fundamental, elemental description of information. These rules can be applied and used across all computing and informational systems.</p>
DTD	<p>This is an acronym for Document Type Definition. This is a document that provides rules that a designer adds to extend the core rules of XML syntax.</p> <p>This is one of the mechanisms by which self-definition or self-description is added to the XML protocol.</p> <p>DTDs contain the detail information about what markings that are placed within an XML document mean in terms of data organization and relationships. The XML document itself merely contains the markings and associated data.</p>
DTD/Schema/DTD-containing-XML	<p>This term is used to express the loose equivalent of the DTD method of expressing XML message structure and the Schema method. DTD-containing-XML are XML message documents which also contain DTD information.</p> <p>With XML, DTDs can be embedded within an XML document.</p>
Schema	<p>Schema is a Microsoft proposal (RFC) to the W3C in an attempt to overcome various limitations of DTDs. Schema is a set of information (document) that provides rules which a designer adds to extend the core rules of XML syntax.</p> <p>This is one of the mechanisms by which self-definition or self-description is added to the XML protocol.</p> <p>A Schema contains detail information about what markings that are placed within an XML document mean in terms of data organization and relationships. The XML document itself contains only the markings and associated data.</p>

Term	Description
XML schema	This is an XML document that uses specialized markups to specify the structure of another XML document and the constraints upon that document's content.
OCM	<p>This is an acronym for Our Content Model. This is used to designate files written in a system proprietary language that is used to express the hierarchical structure of an XML message.</p> <p>OCM files are compiled from DTD, XML Schema, or DTD-containing-XML files. This single compilation step of OCM files reduces overhead processing. This would otherwise be required to provide message-structure information to different parts of the system.</p>
Package	This is a system file structure construct used to organize user DTD/Schema/DTD-containing-XML. This provides the user with a mechanism that keeps organized potentially complex files.

User operations

It is your responsibility to clean up unnecessary files on the server.

Each package folder is a location that is filled by the user with at least one package root DTD/Schema/DTD-containing-XML file. All necessary external support DTD/Schema/DTD-containing-XML files for that package root must be in system-accessible locations.

A package can contain numerous package root and associated support DTD/Schema/DTD-containing-XML files. The XML Package Manager does not differentiate between these file types. It is your responsibility to distinguish package root from external reference support files.

All package root DTD/Schema/DTD-containing-XML files must be immediately under the package folder level. You can create sub-folders to organize package root support files.

At several points in GUI operation, the system-services of the Java language check the storage drive units on the client and server computers. This might invoke a dialog box advising that a disk is missing in a removable drive unit. If this happens, then clear the dialog box by supplying the missing disk to the unit and selecting **Retry/Try Again** or selecting **Ignore/Continue**.

Root names

The DTD root node must match the file name; otherwise, an error happens.

If the file name does not match the root name, then you can compile the DTD using the `hcixmlcompile` command line tool. With this tool, use the `-r rootname` argument to specify the root name.

For XML files containing DTD, the file name does not matter. The XML Package Manager ascertains the correct root name from the XML file.

For Schema files, the file name does not matter. The XML Package Manager compiles all possible messages that are defined in the Schema. If you must specify a single root, then use the `hcixmlcompile -r rootname` command line argument.

XML Package Manager security

The XML Package Manager interfaces with the extant security features of the system, including security server (advanced).

The package division is the foundation for security server (advanced) fine-grain access control to the information that is managed by the XML Package Manager.

Permission for can be granted to individual users for individual packages:

- `READ`. This is permission to copy.
- `WRITE`
- `DELETE`. This is permission to delete.)
- `INSERT`. This is permission to create or paste new packages/folders. . This is permission to compile, paste, rename, or create non-package folders.

This table shows the required permissions for menu options:

Menu options	Permissions
File > New	New package creation requires <code>INSERT</code> permission. Other folder or sub-folder creation requires <code>WRITE</code> permission.
File > Delete	This requires <code>DELETE</code> permission.
File > Rename	Package rename requires both <code>DELETE</code> . This is permission to compile, paste, rename, or create and <code>WRITE</code> permission.
Edit > Copy	This requires <code>READ</code> permission.
Edit > Cut	This requires <code>READ</code> , <code>WRITE</code> , <code>INSERT</code> , and <code>DELETE</code> permissions.
Edit > Paste	Within packages, Paste requires <code>WRITE</code> permission. In the <code>formats/XML/</code> directory at the package level, Paste requires <code>INSERT</code> permission.
Compile > Compile	This requires <code>WRITE</code> permission.

XML definition files

These standard XML definition files are supported:

- Document Type Definition (.dtd)
- DTD-containing-XML (.xml)
- XML Schema Definition (.xsd)

XML files reside in `$HCISITEDIR/formats/`.

These files, which are text documents, use a syntax consisting of programmatic statements and key words that relate to the structure of an XML message. These files provide the XML standard means against which XML messages are checked for validity. Because they are text, they may be written using any text editor.

The schema files must be well-formed and valid according to their standard specifications before they are compiled in the system.

DTD and DTD-containing-XML

DTD is the oldest and most primitive schema format for XML. It has no support for features such as namespaces.

DTD-containing-XML are XML message documents that also contain DTD information.

Schema

XSD is a later definition language than DTD and is more powerful than DTD in describing XML languages.

See <http://www.w3.org/TR/xmlschema-0/>

Most of the grammar and semantics described in the XML Schema specification is supported, with these exceptions:

- With the XML Schema definition, you can constrain the values in an element or attribute through user-defined data types. It provides a list of primitive types with facets that you can use to limit the set of acceptable values. Some of these primitive types are supported, but for more complex types the values must be within the constraints. Any type that does not convert to a system supported type is left as a string.
- An XML Schema document can internally contain references to other external DTD/Schema/DTD-containing-XMLs that are required for root file processing. For example, using the import or include tag. It is your responsibility to edit external references so that they remain within these specified confines. You must also maintain those external resources and assure that they exist within the server.

File management

XML definitions should be deployed using a package folder management scheme. This scheme must provide a flexible, organized style that maintains the file and folder structure that is required by the system operations. For example, Translation Configurator or security server.

XML processing requires the placement of DTD/Schema/DTD-containing-XML files in a specific directory location on the server. This location is `/HCIRoot/integrator/site/formats/xml/`.

`HCIRoot` and `site` are the system installation root and site.

You can organize working DTD/Schema/DTD-containing-XML files into package sub-folders immediately under this specific folder.

For example, `/HCIRoot/integrator/site/formats/xml/package name`.

`package name` is a user-provided package folder name and `HCIRoot` and `site` are the system installation root and site.

You can also create complex structures on the server using this scheme:

```
"package root"/package name/someroootDTD
"package root"/package name/anotherrootDTD
"package root"/package name/anotherrootDTD
"package root"/package name/supportfolder/supportDTD
"package root"/package name/supportfolder/another supportDTD
```

"package root" is `.../HCIRoot/integrator/site/formats/xml/`.

You can create any depth of sub-folders within package folders to organize support files. These sub-folders cannot contain package root DTD/Schema/DTD-containing-XML files that are to be compiled.

Package folders

The system built-in implementation of XML handling requires that the user pre-define the structure of any message that processes using DTD/Schema/DTD-containing-XML. The system further requires storing these defining DTD/Schema/DTD-containing-XML files in an organized manner upon the server.

The package folder is the primary container in which users may organize and maintain the DTD/Schema/DTD-containing-XML files on the server. These files may not be placed upon the server without being contained within a package folder.

The Client File Viewer pane shows an unrestricted view of the file storage structure on the system machine.

The server Package Viewer pane shows the file storage structure on the server machine. For security and processing purposes, this view is restricted to and defaults to a package root folder `xml`. The location of package root on the server is `/HCIRoot/integrator/site/formats/xml/`.

Within this folder, you can create special sub-folders called packages, which:

- Organize package root DTD/Schema/DTD-containing-XML files into locations expected by other processes within the system.
- Organize any externally-referenced support files.

Externally-referenced file placement

XML file management provides flexibility for user-placement of any non-Internet support files, or externally-referenced files, for a package root DTD/Schema/DTD-containing-XML file.

Options are available for these files, where you can:

- Place them directly in the package containing the package root.

- Organize them in user-created sub-folders of the root's package folder.
- Place them into a separate package folder library.

For external references, pre-edit the external references in the package root DTD/Schema/DTD-containing-XML files to reflect the user-created structure.

Managing XML configuration files

The XML Package Manager provides users with the means to create and view package folders upon the server machine. It also provides users with file maintenance tools. With these tools, you can transfer files into their packages on the server. You can also maintain these files through the editing actions.

The XML Package Manager can be thought of as two Windows-type file explorers combined into one tool. Each file explorer presents a file-tree view of portions of a computer or computers to which the user is granted access. These views are shown side-by-side.

- On the left side of the GUI is a view of the file system on the client machine.
- On the right side of the GUI is a view of a portion of the file system on the server machine.

In some installations, the client and server machines are the same. For this, you can adjust the left view to show the same file area as the right side view.

Select items in the left side view for copy, cut, or paste to the right side. Do this by moving files from the client to the server. Select items to perform the reverse transfer, or to otherwise move and maintain files on the client or the server.

The DTD/Schema/DTD-containing-XML root files and local resource files must be maintained in a package structure on the server machine.

Root DTDs must be placed within the top-level package folder. Only DTDs that can be compiled are under the package folder.

This rigidity is required so that the Translation Configurator and translation engine can readily find required input files. The package structure underlies the fine-grain access control of security server (advanced). This package scheme also encourages users to keep potentially confusing groups of message structure definition files in a manageable organization.

The XML Package Manager is the tool by which users organize their DTD/Schema/DTD-containing-XMLs into the required package structure on the server. Additionally, the engine and other GUI components require a pre-compile of the DTD/Schema/DTD-containing-XML. The XML Package Manager provides this compilation.

File and folder organization

Remember these points when working with files and folders:

- The files and folders in each pane are organized alphabetically. Folders containing child folders or files are marked with a plus sign (+).
- Tree items at each branch are shown in the order: folders first and then files.
- Directories are shown above files in the tree. This higher location does not necessarily indicate that the files below a sub-folder are its children.

- The simultaneous views of the client and server file structures enable visual file management.

In normal work flow, gather files in the Client File Viewer pane. This pane's file tree serves as a means by which you can locate created files and folders. Then, use the XML Package Manager to transfer files that were gathered on the client machine into package locations on the server machine. This is the Server Package Viewer pane.

Note: Exercise care when naming files and folders, because packages, folders, and files can legally have the same name.

Points to remember:

- You can select any number of file and folder items at one time. File and folder items are selectable in the server tree or client tree.
- Simultaneous item selection, from both the server and client trees, is not permitted.
- Folder selection implies all contained child sub-folders and files.

Restrictions

XML Package Manager restrictions include:

- Folders are not permitted as children of files.
- There are no restrictions on depth of folder nesting.
- Folders cannot be created or renamed when multiple parent targets are selected.
- Renamed or new folders cannot have the same name as a pre-existing folder at the same tree level.
- Folders cannot be created or renamed using a name inconsistent with the rules of the underlying computer file system.
- The visual label roots of the client or server file trees. For example, My Computer or the host server cannot be selected as valid copy objects.
- Before a paste is completed, a warning displays if the paste overwrites an existing file or folder.
- Files are not permitted as direct children of the `... /xml/package root` folder in the Server Package Viewer pane. Only folders may be pasted to this root.

In the Client File Viewer pane, files can be pasted to the `/HCIRoot/integrator/<site>/formats/xml/` directory if the client and host are installed on the same machine.

- File system root deletion is not permitted (for example, users cannot delete a client drive designator or the `.../formats/xml/` directory). The action is disabled for this selection.
- You cannot rename file system roots. For example, you cannot rename a drive designator or the `/HCIRoot/integrator/formats/xml/` directory in the Server Package Viewer pane.

The Rename action is disabled for the `/xml/package` root directory in the Server Package Viewer pane. In the Client File Viewer pane, files can be renamed if the client and host are installed on the same machine.

- Renaming a folder can have deleterious effects upon XML file relationships, especially when external references are present within package/folder member files. A notification and opportunity to abandon is given at each rename.
- Security Server (advanced) package Rename requires both `DELETE` and `WRITE` permissions.
- Security Server (advanced) requires these permissions:
 - Copy requires `READ` permission.
 - New package creation requires `INSERT` permission.
 - Other folder or sub-folder creation requires `WRITE` permission.

- Cut requires READ, WRITE, INSERT, and DELETE permissions.
- Deletion requires DELETE permission.
- Within packages, Paste requires WRITE permission.
- In the `formats/XML/` directory at the package level, Paste requires INSERT permission.

See [Security](#).

Creating a new folder

- 1 Select the file system folder, client or server machine, in which to place the new folder.
- 2 Click **New Folder** on the toolbar. This opens the **Create New Folder** dialog box.
- 3 Specify the **New Folder Name** and select a **Folder Type**.
For CDA folders, select a **Version** and **Type**.
Note: **Folder Type** and **Options** are available only when creating a new folder in the Server Package Viewer pane. Folders that are created in the Client File Viewer pane only require a **New Folder Name**.
- 4 Click **OK**. A child folder is placed under the selected file system folder.

Renaming files/folders

- 1 Select the file or folder to rename.
- 2 Select **File > Rename** on the menu bar, or click the toolbar button. This opens a confirmation dialog box.
- 3 To continue, click **Yes**. This opens the **Rename File/Folder** dialog box.
- 4 Specify the new name. Any name is permissible, as long as it conforms to the file system on the client or server machine.
- 5 Click **OK**.

Compiling/Recompiling

DTD/Schema/DTD-Containing-XML definition files must be compiled into OCM files before they can be used in the system.

Each XML definition file is compiled into a system-specific internal file called our content model files, designated by the `.ocm` extension. This file is placed in the same folder as the root of the XML

DTD/Schema/DTD-containing-XML that was compiled. These `.ocm` files are for internal purposes only and are not intended to be user-edited.

During the compilation process, the compiler resolves all external references from the root.

Although they exist in the same folder as the compiled file, they are not shown in the Package Manager's view of the containing package folder. Files with an `.ocm` extension may be shown at other file-tree folder levels.

Each successfully created OCM file is a system expression of the structure of an XML message originally described by the original DTD/Schema/DTD-containing-XML.

An OCM file is necessary to relay that structure to the parts of the system that require the information, such as the Translation Configurator.

An OCM file must be created before you can select that format for an XML message.

Note: Tampering with OCM files can cause your XML implementations to become unstable. OCM update does not automatically happen if a DTD/Schema/DTD-containing-XML should change. If one of the original files defining the structure of an XML message changes, then use the **Compile** option in the XML Package Manager. You can also use `hcxmlcompile` on the command line to update the associated OCM.

In addition to OCM files, an `ocmIndex` file is generated. This keeps track of the relationship between the OCM files and the original DTD or Schema files. Each package directory has an index file.

To compile, you can use the `hcxmlcompile` command-line tool or the **Compile** option in the XML Package manager.

- In the XML Package Manager, select a file/folder and click **Compile** on the toolbar or from the right-click menu.

It is also possible to specify a root element and output name as at the command line. This is available only if a single file is selected when **Compile** is clicked. In this case, when **Compile** is clicked, the **Compile Options** dialog box opens.

- From the command line, run `hcxmlcompile`.
See [hcxmlcompile](#).

Both methods invoke `hcxmlcompile` and pass to it the package name and file name only. You can also select multiple files and compile them at one time.

When compiling a Schema that contains more than one global element definition, the `-r` option must be used with `-o`. This specifies which root element to use.

hcxmlcompile usage

This command line is internally-generated and launched into the host server's processing environment by the XML Package Manager. This amounts to the launch of a process monitor that monitors output on the system server's stdout and stderr. Any output on stdout or stderr in response to the command-line instruction constitutes an exception. This exception is shown by an advisory dialog box that presents the stdout or stderr content, listing the input file that failed OCM creation.

Commands are cached as they are specified. The process monitor returns the command string that launched the process as each process completes. Each command string returned is removed from the cache upon return. When the cache reaches zero content, the XML Package Manager tree update is triggered. This updates the visualization of successfully-compiled files, signified by green check marks, upon the tree.

The `hcxmlcompile` command updates the OCM file that is associated with a DTD/Schema/DTD-containing-XML file that has changed. It analyzes the structure of the DTD/Schema/DTD-containing-XML, and creates an object model of the structure used to process and map the XML within translation.

Two arguments, `-r` and `-l`, are provided for specifying the root element for which to generate an OCM and list all the global element declarations.

See [hcxmlcompile](#).

By default, all possible roots that are defined in the XML definition file are compiled. If the `-r` option is specified during compile, then the OCM file is only created for the specified root.

Default namespace for encoded XML

In an XML instance document that uses namespaces, there is an option to define a default namespace. In this way, all unqualified XML elements in the XML document are processed as being qualified by this default namespace.

For example:

Without the default namespace declared:

```
<?xml version="1.0"?>
<apo:purchaseOrder xmlns:apo="http://www.example.com/P01"
                    orderDate="1999-10-20">
    <apo:shipTo country="US">
        <apo:name>Alice Smith</apo:name>
        <apo:street>123 Maple Street</apo:street>
        <!-- etc. -->
    </apo:shipTo>
    <apo:billTo country="US">
        <apo:name>Robert Smith</apo:name>
        <apo:street>8 Oak Avenue</apo:street>
        <!-- etc. -->
    </apo:billTo><apo:comment>Hurry, my lawn is going wild</apo:comment>
    <!-- etc. -->
</apo:purchaseOrder>
```

With the default namespace declared:

```
<?xml version="1.0"?>
<purchaseOrder xmlns="http://www.example.com/P01"
               orderDate="1999-10-20">
    <shipTo country="US">
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <!-- etc. -->
    </shipTo>
    <billTo country="US">
        <name>Robert Smith</name>
        <street>8 Oak Avenue</street>
        <!-- etc. -->
    </billTo>
    <comment>Hurry, my lawn is going wild</comment>
    <!-- etc. -->
</purchaseOrder>
```

Pretty print encoding format

For readability and debugging, you can specify how to format the target XML message before it is validated.

Select **Format Encoded XML** on the **Compile Options** dialog box to use pretty print. This adds `-F pretty` to the dialog box command line.

For `hcixlttest` and `hcxmltest`, use the `-l` option for pretty print.

Example output:

Current target XML message:

```
<a><b><c>val</c></b></a>
```

Pretty-printed target XML message:

```
<a>
  <b>
    <c>    val
    </c>
  </b>
</a>
```

Default behavior

The default behavior, without the `-r` or `-l` arguments, includes:

- DTD files.
The file name is used to identify the root name. If there is no requirement to change the DTD name, then you must use the `-r` option to specify the root name.
Only one OCM file is generated from a DTD at a time. If multiple messages are defined by a single DTD, then you must use the `-r` option from the command line.
- XML files.
If an XML file is used, then it must contain an embedded DTD or a reference to an external DTD or Schema. The root element of the XML document is used to identify the message name.
The `-r` option is ignored for XML files because usually you do not specify a different root from what is in the XML file.
- Schema files.
A single OCM file is created for each schema file. If an error happens, then no OCM file is created. This schema file contains information for all global element declarations in the schema. When selecting this OCM file to be used in a translation, you must select which root element to use. The name of this OCM file is the same as the schema file name.
To create an OCM file for a single root, use the `-r` option from the command line. This creates an OCM file that is named after the root.

- If there is a target namespace, then the file name is named:

```
target namespace$root name.ocm
```

- If there is no target namespace, then the file name is named:

```
root name.ocm
```

Examples

Using the `hcixmlcompile` option to use a default namespace for encoding XML is set at compile time.

- To let the XML compiler determine the default namespace:

```
hcixmlcompile <current_compile_args>
```

- To explicitly set no default namespace to replicate current behavior:

```
hcixmlcompile -N none current_compile_args
```

- To set the default namespace to the target namespace:

```
hcixmlcompile -N target current_compile_args
```

- To set the default namespace to another namespace:

```
hcixmlcompile -N "http://www.othersns.com" <current_compile_args>
```

Using the `hcixmlcompile` option to change the white space for the encoded XML is set at compile time.

- To implicitly have the target XML message encoded on a single line as it was before:

```
hcixmlcompile current_compile_args
```

- To explicitly have the target XML message encoded on a single line as it was before:

```
hcixmlcompile -F none current_compile_args
```

- To have the target XML message encoded with newlines between elements and tabbed whitespace to denote nesting level:

```
hcixmlcompile -F pretty current_compile_args
```

Specifying the name of the output OCM file

The OCM file can have any name.

When running `hcixmlcompile`, there is an `-o` option for specifying the name of the output OCM file.

```
hcixmlcompile -f filename -p package dir [-r root element name] [-o  
output file name] [-v on/off] [-d on/off] [-n on/off]  
[-e on/off] [-L on/off] [-F pretty/none] [-N /target/namespace URI/none]  
[-b] [-l] [-c] [-s] [-h]
```

When using this command:

- *-r root element name* only creates an OCM for this message definition.
- *-o output file name* is the name of the output message definition.

If the *-o* option is not used, then the OCM file is named after the root element name. If a name is specified, then the file is named as specified with the *.ocm* extension, if the specified name does not already include it.

Compiling from the GUI

Select the file within package folders to compile, where each selected file is a valid DTD/Schema/DTD-containing-XML and click **Compile**.

This opens the **Compile Options** dialog box.

See [Compile Options dialog box](#).

This action consists of an analysis of each selected file. The selected file is a valid root DTD/Schema/DTD-containing-XML. This is if the analysis can resolve any external references and compile the message structure information contained therein.

This option is available only when a file is selected within a package folder. If a folder or non-package file is selected, then the menu bar and toolbar option are unavailable.

This action is available when a mixture of valid and invalid targets is selected. Only the valid selections are compiled.

Recompiling

To recompile a file with the previous compile options, select **Compile > Recompile** from the menu bar. You can also right-click in the Server Package Viewer pane to open the menu.

You can also simultaneously recompile multiple files by multiple selections. An error message box opens if an error happens.

The **Recompile** option is enabled only on a compiled XSD/DTD file. This option is unavailable for a schema or DTD that is not compiled, but is enabled for multiple selections. This option is also enabled on a package and on a folder/sub-folder.

When you select a folder and click **Recompile**, all the XSD/DTD files under the folder, the sub-folders, or the sub-sub-folders are recompiled.

Compile is enabled when the selection item is a folder. The running of compile follows the same rules as recompile.

Empty node pruning

Specify the *-e* option to disable pruning. For example, if you copy to a repetition but a previous repetition has not been created, an empty node is created by the engine. This assigns the value to the correct repetition. The engine prunes these by default.

For example:

```
COPY =x -> root(0).child(2).#text
```

This statement creates `root(0).child(1).#text` as an empty node.

Content model (OCM) files

The Compile action of the XML Package Manager creates a content model file designated by the `.ocm` extension. This file is placed in the same folder as the root of the XML DTD/Schema/DTD-containing-XML that was compiled. These `.ocm` files are for internal purpose only and are not intended to be user-edited.

The content model compiler/tester is a portion of the system that compiles a content model from a root DTD/Schema/DTD-containing-XML file. In the process, this compiler/tester resolves references to external references.

HL7 v3 CDA 2.0 support

Note: For additional information on CDA 2.0, visit <http://www.hl7.org>.

The HL7 Version 3 CDA (Clinical Document Architecture) Framework Release 2.0 is a standard. This standard defines the semantics of condition and the structure of clinical documents when exchanged by HL7 messages. It is a standard for exchanging HL7 messages when the incoming message format is XML and the internal structure of the message is defined by HL7 v3 CDA.

The CDA provides an exchange model for clinical documents, such as discharge summaries and progress notes.

Key aspects of the CDA include:

- CDA documents are encoded in Extensible Markup Language (XML). By leveraging the use of XML, the HL7 Reference Information Model (RIM) and coded vocabularies, the CDA makes documents both machine-readable and human-readable. CDA documents can be displayed using XML-aware web browsers or wireless applications such as cell phones.
- CDA documents derive their machine-processable meaning from the HL7 Reference Information Model (RIM) and use the HL7 Version 3 data types.

CDA specifies the structure and semantics of clinical documents for the purpose of exchange. The scope has expanded through usage. For example, some implementations are using CDA to exchange laboratory reports or prescriptions. A common question relates to the distinction between an HL7 document and an HL7 message, and knowing which to use when.

Although there are gray zones, messages tend to be transient, trigger based, and nonpersistent. Clinical documents have persistence, wholeness, clinician authentication, and are human readable.

CDA document structure

A CDA document is wrapped by the `<ClinicalDocument>` element, and contains a header and a body. The header lies between the `<ClinicalDocument>` and the `<structuredBody>` elements. This identifies and classifies the document and provides information on authentication, the encounter, the patient, and the involved providers.

The body contains the clinical report. This is an unstructured blob or contains structured markup. This example shows a structured body, which is wrapped by the `<structuredBody>` element, and which is divided up into recursively nestable document sections.

A CDA document section is wrapped by the `<section>` element. Each section can contain a single narrative block, and any number of CDA entries and external references.

The CDA narrative block is wrapped by the `<text>` element within the `<section>` element, and must contain the human-readable content to be rendered.

Within a document section, the narrative block represents content to be rendered. CDA entries represent structured content provided for further computer processing. For example, decision support applications. CDA entries typically encode content present in the narrative block of the same section.

The example shows two `<observation>` CDA entries, and a `<substanceAdministration>` entry containing a nested `<supply>` entry, although several other CDA entries are defined.

CDA entries can nest and can reference external objects. CDA external references always happen within the context of a CDA entry. External references refer to content that exists outside this CDA document. For example, an image, procedure, or observation that is wrapped by the `<externalObservation>` element. Externally referenced material is not covered by the authentication of the document referencing it.

Although the narrative blocks must always be present, the CDA entries are optional. An originator of a CDA document is not required to fully encode all narrative into CDA entries within the CDA body. A recipient is not required to parse and interpret the complete set of CDA entries contained within the CDA body.

Within an implementation, trading partners can ascribe additional originator and recipient responsibilities. These responsibilities create various entries and can create various templates or implementation guides that require the use of various entries.

Therefore, CDA can be implemented or relatively detailed to implement. This provides a migration pathway toward progressively richer computer-processable content.

Example:

```
<ClinicalDocument>
... CDA Header ...
<structuredBody>
  <section>
    <text>...</text>
    <observation>...</observation>
    <substanceAdministration>
      <supply>...</supply>
    </substanceAdministration>
    <observation>
      <externalObservation>...
    </externalObservation>
    </observation>
  </section>
  <section>
    <section>...</section>
  </section>
```

```
</structuredBody>  
</ClinicalDocument>
```

Using the schemas

To use these schemas in a site, import the schemas from the root using the XML Package Manager.

The reference XML document definitions are provided in these XML packages located at `$HCIRoot/formats/xml`:

- HL7_CDA_1.0-dtd
- HL7_CDA_1.0-xsd

These are pre-compiled and work without compilation after they are imported.

Creating a new CDA package/folder

- 1 Select **New Package/Folder** from the **File** menu or by right-clicking in the Server Package Viewer pane and selecting **New Package/Folder** from the context menu.
This opens the **Create New Folder** dialog box.
- 2 Specify a folder name and select HL7 CDA from the **Folder Type** list.
- 3 Select the **Version** and **Type**.
- 4 Click **OK**. A new folder is created with pre-compiled CDA XSD files or CDA DTD files.

Compile Options dialog box

To access the **Compile Options** dialog box, right-click an `.xsd` node and select **Compile**, or click the **Compile** tool.

This table shows the available options:

Option	Description
Existing XML	Indicates how many XMLs to which the current schema/DTD are compiled. When you select one item, all the compiler options are set to keep the same as those used in the selected compiling. If the schema or DTD was never compiled, then this option is not available.
Root Element	Specifies the root element name. This is passed as the <code>-r</code> option to <code>hcxmlcompile</code> .

Option	Description
Output Name	Specifies the name of the message definition (OCM file). This is passed as the <code>-o</code> option to <code>hcxmlcompile</code> . This can be set even if the Root Element option is not selected. Note: The default is to not pass the <code>-r</code> or <code>-o</code> flags to <code>hcxmlcompile</code> .
Format Encoded XML	Select for "pretty print" (legible) output. This is for debugging. Select one of the options in the Default Namespace pane to set the default namespace for encoding XML. The default is Use the Default Namespace in XSD File .
Use Target Namespace	Sets the target namespace as the default namespace.
Use Namespace	Specifies the namespace in the text field.
Command	Shows the command in the compilation pane. You can copy and paste it on the command line, as with the Database Administrator and the Testing Tool.

XML Compile Properties dialog box

To see how a schema or DTD file was compiled, select **Compile > Properties** from the menu bar. You can also right-click in the Server Package Viewer pane and select **Properties**.

This opens the **XML Compile Properties** dialog box.

Note: **Properties** is only enabled on a compiled XSD/DTD file. This option is unavailable for a schema or DTD that is not compiled or with multiple selections.

Schema support

As with DTDs, schema describes the structure of an XML document.

Note: The path syntax for schema messages is the same as DTD messages.

Additionally, schema describes the data types of the values. Although DTD has its own format, a schema document is formatted in XML.

For schema files, the file name does not matter. The XML Package Manager compiles all possible messages that are defined in the schema. If you must specify a single root, then use `hcxmlcompile -r rootname`.

See [hcxmlcompile usage](#).

Testing Tool output

For `hcixlttest` output, content matching the wildcard is printed with the wildcard address.

For example, the input is:

```
<root xmlns="primaryns" xmlns:ns1="otherns">
  <child>normalchild</child>
  <ns1:otherchild>
    <ns1:mywildcard/>
  </ns1:otherchild>
</root>
```

The schema used is:

```
<xsd:element name="root">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="child"/>
      <xsd:any namespace="##other"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

The testing output is similar to:

```
Command output:
MESSAGE 1
root.child.#text:    >normalchild<
root.##any:         ><ns1:otherchild><ns1:mywildcard/><ns1:otherchild><
```

Note: The XML string that is matched to a wildcard explicitly contains namespace declarations for all namespaces used by elements and attributes inside the wildcard XML. In this way, `xmlns` attributes may be inserted if namespaces used by the wildcard elements are originally declared in an upper level. With this, any users or systems can determine all namespace references by looking at the XML string that is matched to the wildcard.

In the previous example, the namespace declaration is:

```
xmlns:ns1="otherns"
```

This is added to:

```
root.##any
```

The result is that all namespace mappings in this XML string are fully defined:

```
<ns1:otherchild xmlns:ns1="otherns"><ns1:mywildcard/><ns1:otherchild>
```

Namespace support

XML namespaces provide a way to differentiate between elements with the same name.

An XML message is identified by the name of the root node. To accommodate namespaces, messages are identified by the namespace URL and the root node name. If a message does not specify a namespace, then the message only uses the root node name.

If a namespace is specified, then the TrxID is the namespace URL and the root node name separated by a colon.

If no namespace is specified, then the root node name is returned.

A namespace is declared in an XML document with a namespace attribute. This attribute can be `xmlns` or `xmlns:name`, where `name` is used to reference the namespace. The value of the attribute is a unique URL.

The system can identify an element or attribute by its namespace URL. Any namespaces that are defined in a message are stored in the OCM for reference. Representations of a namespace-qualified element or attribute have the namespace prefix that was imported during the generation of the OCM file.

When an XML message is parsed, the qualified name of an element or attribute is used to match against the OCM tree. The qualified name is the namespace URL and local name.

When an XML message is encoded, elements and attributes are namespace qualified automatically based on the information in the OCM file.

OCM files and namespaces

The OCM file is created by `hcxmlcompile` from a DTD, an XML document containing a DTD, or a schema document.

See [Root names](#) and [hcxmlcompile usage](#).

The OCM file has a `namespace=` section that is a mapping of namespace URL to namespace prefix. A prefix is generated for each namespace that does not have a predefined prefix. The namespaces that display in this section come from the target namespaces declared in the base schema and any schemas that are imported.

Examples:

- RootA schema is compiled that has a target namespace of `http://www.targeturl.com`. This schema imports two other schemas with target namespaces of `http://www.schema1targeturl.com` and `http://www.schema2targeturl.com/purchaseOrder`. The first two schemas do not specify a prefix for the target namespace but the third one does.

The namespace section of the OCM would be:

```
namespace=nm1 http://www.targeturl.com
nm2 http://www.schema1targeturl.com
po http://www.schema2targeturl.com/purchaseOrder
```

The prefixes `nm1` and `nm2` are generated by `hcxmlcompile` because they are not defined in the schema. The last prefix, `po`, was defined in the schema, so that prefix is used.

- The OCM tree section describes the structure of the message. The elements and attributes that are associated with a namespace have the prefix added to the name.

```
nm1:foo {1,1} (,)
    nm2:bar {1,1} (|)
        #text (str)
```

The `nm1` prefix associates the element `foo` with the namespace URL `http://www.targeturl.com` and the `nm2` prefix associates `bar` with the namespace URL `http://www.schema1target.com`.

This prefix is also used in the path for this element, although the actual message being parsed may use a different prefix. When a message is encoded, the prefix from the OCM is used.

- If the root element is associated with a namespace, then the name of the file must reflect the namespace and the root name.

Currently, a DTD-generated OCM file that contains a prefixed root element has a file name with the prefix and element name with a `$` delimiter. For a Schema generated OCM, the prefix is machine generated. It rarely matches the actual message, and is likely to collide with another OCM file. To solve this problem, the namespace URL of the root element is used.

A URL can contain characters that are not legal in a file name, so some substitutions are made. As with DTD-generated OCM files:

- Colons (`:`) are replaced with dollar signs (`$`)
- Slashes, forward or backward, are replaced with percent signs (`%`)

In this example, the root element is `foo` and it is associated with the namespace `http://www.example.com/test`.

The file name is `http$%%www.example.com%test$foo`.

Note: The maximum file name length is 215 characters, because this is the maximum length accepted by Windows. If the file name is longer than 215 characters, then the namespace URL is truncated to fit.

This example is true only if `-r` is used, where you specify the root element for which to generate an OCM.

Transaction ID determination

The TrxID of an XML message is the name of the root element. For messages with no namespace that is associated with the root element, the TrxID is the root element. For messages with a namespace, the TrxID is one of two possibilities:

- The namespace prefix followed by the element name.
- The namespace URL followed by the element name.

Which one is used depends on whether the message definition was generated from a DTD or a schema:

- If it is based on a DTD, then the TrxID is the namespace prefix followed by the element name.
- If the OCM was created from a schema, then the TrxID is the namespace URL followed by the element name.

In the definition of a protocol thread, there is a `DATAFORMAT` key in the Netconfig file that contains a `TYPE` key.

XML TrxID determination uses `xml` as the value for this key.

XML (Namespace) TrxID determination uses `xml_ns` as the value for this key. This value uses the namespace URL instead of the namespace prefix.

Specify the correct value based on the message formats. This is configured on the Network Configurator's property panel.

Examples:

- The root element is prefixed associating it with a namespace:

```
<cis:foo xmlns:cis='http://www.example.com/test'>
```

- TrxID for `xml`:
`cis:foo`
- TrxID for `xml_ns`:
`http://www.example.com/test:foo`

- The root element is not prefixed, but belongs to a default namespace:

```
<foo xmlns='http://www.example.com/test'>
```

- TrxID for `xml`:
`foo`
- TrxID for `xml_ns`:
`http://www.example.com/test:foo`

- The root element does not belong to a namespace:

```
<foo>
```

- TrxID for `xml`:
`foo`
- TrxID for `xml_ns`:
`foo`

Moving an XSD file to host and compiling

To move an XSD file from the client to host and then compile the XSD:

- 1 Transfer (copy/paste) the XSD file generated on your local machine (Client File Viewer) to the host server (Server Package Viewer pane).
- 2 After the XSD files are transferred, right-click an `xsd` file in the server side tree and select **Compile**. This opens the **Compile Options** dialog box.
- 3 Select options as necessary.
See [Compile Options dialog box](#).

Index files

The relationship between the OCM files and the original DTD or schema files is kept in an index file. Each package directory has an index file.

Lines that begin with # are comments and are ignored.

The source DTD, XML, or schema file name is on a line by itself, followed by the generated OCM entries.

Each OCM entry contains the root name followed by the OCM file name. Both are enclosed in curly braces.

Example:

```
#This index file is used by the XML Package Manager GUI
#It is maintained by hcxmlcompile and should not be edited manually.
#Removing or editing this file will prevent the GUI from displaying
#Information properly.
levelone_1.0.xsd
{levelone}{levelone.ocm}
```

The index file is updated by `hcxmlcompile`.

For DTD files, if an entry for the OCM being generated already exists in the index file, then it is replaced. If there is no entry for the OCM or the original DTD, then it is created.

Because an XML file can only specify one root element, only one OCM file can be generated. This OCM entry is created if one is not present or replaced if one already exists.

For schema files, the default behavior of not specifying a root element deletes all existing OCM entries for this schema. It then adds all new entries. If a root element is specified, then that entry alone is replaced.

Testing tools

The testing tools are used to build and test a system solution. You build the components of a system before testing them, and then integrate them into larger components.

These tools perform selected tests by generating appropriate command-line commands along with the correctly ordered requisite parameters.

Use the system tools to perform unit, integration, and system testing on your network configuration.

Testing tools are accessed through the GUI or the command-line.

See each of these Testing tool topics for command-line commands and GUI fields:

- Database protocol test: This calls the engine command line to test the specified database protocol thread. See [Testing the database protocol](#) on page 886.
- Database schema test: This parses the database schema format. See [Testing the database schema](#) on page 887.
- DICOM test: This runs a DICOM ECHO test, by doing a DICOM ping action on the DICOM remote SCP (Service Class Provider). This is to verify the DICOM network between the system DICOM SCU and the remote DICOM SCP. See [Testing DICOM configurations](#) on page 888.
- JSON test: The JSON data message file is based on the specified JSON schema file. You can select from the current site, master site, and root-level JSON schema files. See [Testing JSON configurations](#) on page 893.
- LDL test: LDL (Length Delimited Layout) is a data layout used for DICOM. Functionality is similar to FRL and VRL. See [Testing LDL configurations](#) on page 894.
- Record format test: These tools test configurations by simulating the input-parse portion of the translation engine. These tools read, parse, and display messages, ensuring that configurations match the input data. For example, the FRL tool tests one particular FRL (Fixed Record Layout) for all of its input messages. It then test compiles an FRL to ensure the engine can use it. Using the Testing tool for other record formats is more complex, as other formats encompass entire version or variant sets, which contain many message definitions.
 - See [Testing EDIFACT configurations](#) on page 889.
 - See [Testing FRL configurations](#) on page 890.
 - See [Testing HL7 configurations](#) on page 890.
 - See [Testing HPRIM configurations](#) on page 891.
 - See [Testing HRL configurations](#) on page 892.

- See [Testing NCPDP Telecom configurations](#) on page 902.
- See [Testing NCPDP SCRIPT configurations](#) on page 903.
- See [Testing NCPDP F&B configurations](#) on page 904.
- See [Testing VRL configurations](#) on page 900.
- See [Testing X12 configurations](#) on page 900.
- Route tests: This simulates most of the translation thread. Given a message, it determines the transaction ID and brings up the appropriate route configuration. This tool outputs messages to a set of files, or to a Tcl end procedure, depending on the end processing configuration.
See [Testing routes](#) on page 895.
- TCP/IP: Use the TCP/IP Test tool to configure, test, and debug TCP/IP connections.
See [TCP/IP](#) on page 911.
- TPS (Tcl Procedure Stream) tests: This simulates most TPS UPoCs (User Points of Control)
See [Testing TPS](#) on page 897.
- Trx ID test: Tests a transaction ID separately from the engine. This simulates getting the transaction ID on an incoming message from the source thread.
See [Testing a transaction ID](#) on page 899.
- Translation tests: After the record format tests verify the input data, use the XLT tool to define a translation specification that converts translation data into another format.
The only required argument for this test is the name of the translation.
 - If an input file is not supplied, then it checks the translation file for validity.
 - If an input file is supplied, then it internalizes the translation, runs one or more sample input messages through the test.

For every input message, the translation engine produces an output message. The Translation Configurator parses the input message according to the translation's input record specification. Then it displays the results according to the chosen detail level. Given an output file name, the XLT tool writes the translated message to that file.
See [Testing XLT configurations](#) on page 905.
- XML test: Built-in support is provided for XML as a message type. XML provides built-in handling of XML messages, in addition to providing support for the recommendations of the W3C (World Wide Web Consortium). The XML tool takes an XML data file and processes it through the parser to determine if it matches the DTD specifications.
See [Testing XML configurations](#) on page 906.
- XSLT: Tests XSLT files.
See [Testing XSLT configurations](#) on page 907.

Certain test options are only available with certain modes:

- **Choose Data File** and **Line Termination Format** are available only in run mode.
- **Initial Startup** is available in run mode and time mode.
- **Interval** and **Max Messages** are available in time mode.
- When startup mode or shutdown mode are chosen, only **Caller Context** and **Leak Detection** are available.

BLOB/CLOB support in Testing Tool

In the Testing Tool, BLOB/CLOB support is on the **Database Protocol** and **Database Schema** tabs.

Note: The BLOB/CLOB is not supported in the Database Lookup table, CLWizard, or CLAPI.

BLOB/CLOB command line support

The `hcidbprotocolttest` testing tool retrieves the BLOB/CLOB content and saves it into a temp file, located at:

- `current_location/temp`, when running `hcidbprotocolttest` from the command line.
- `$HCIROOT/temp`, when running `hcidbprotocolttest` in the Testing Tool GUI.

The CLOB temp file uses ".clb" as the file extension. This file is UTF-8 encoded.

The BLOB temp file uses ".blb" as the file extension.

The temp file name is composed of these sections:

- A constant beginning with `dbpcache`.
- An intermediate with `sha256sum` of the file.
- An extension.

The Testing Tool retains only one cached temp file (local) if the database table records have the same content for the CLOB/BLOB column.

`hcidbschematest` lists the CLOB/BLOB based on the message's matched fields.

In most instances, the **CLOB** field begins with

"3A45E466D51049A11BCC3E52AD2216DE2E9046C4E98EA7545B60B8617487DE2" followed by a full file path.

The **BLOB** field begins with "92C674350DE16494C901601E1504E96627757E61684CDF28B9A3E2847DE687A", followed by a full file path.

Lookup table extension

All lookup table files must have the `.tbl` extension to be viewed in the Site Manager, Testing Tool, and Translation Configurator. If these tables are created from within the GUI, then they already have the correct `.tbl` extension.

Sites that are brought forward from previous versions that do not use the `.tbl` extension are not required to have the table files renamed to include the `.tbl` extension. They are also not required to have the references updated in the translation that uses the table.

Table lookup files must end in `.tbl` when selecting Lookup Table files in the Translation Configurator and Testing Tool. The Translation Configurator only lists `*.tbl` entries and the Testing Tool gives an error if this is not performed.

File read/writes in the NL mode (newline terminated) do not support the Unicode UTF-16 encoding. The NL mode cannot be performed without knowing the encoding beforehand, which is not known. This applies to all test routines that take a file as input or do file output in the NL mode.

System topology

A reliable system is built using these steps:

- 1 Design the system.
- 2 Configure the system.
- 3 Test the system.

Use this approach several times when configuring and testing your system as an iterative process.

System tests

Use the Testing Tool to test any part of the configuration.

For best results, use all three types of testing in this order: unit, integration, and system.

The Testing Tool performs selected tests by generating appropriate command-line commands along with the correctly ordered requisite parameter in **Preview Command to Issue**.

Start with the unit test.

See [Unit testing](#).

After the unit test, do the integration test, finishing with the system test.

See [Integration and system testing](#).

Unit testing

Unit testing involves constructing and validating the smallest components of a system before integrating them into larger components. To build your record format configuration:

- 1 Use the configuration tools to configure any variants.
- 2 Build TPS modules.
- 3 Use the test drivers to verify record format configurations and variants, such as FRLs, VRLs, and so on, created for the network.
- 4 Verify any Tcl procedures with `hcitptest`.
- 5 Test the record format configuration in the Testing Tool.
- 6 Proceed to the integration and system tests.

See [Integration and system testing](#).

Integration and system testing

Having configured and tested the units with unit testing, integrate them into the network and test them.

- 1 As translations are built, verify the operations before adding advanced operations (for example, procedure calls) to the specification. This approach helps isolate any problems that might arise during construction.
- 2 Verify each translation specification before configuring it into the network. Translation specifications describe how messages from source locations are converted so they can be interpreted by single or multiple destinations. An erroneous translation can result in messages delivered with invalid information.
- 3 Verify translation specifications before adding them to the production engine, and only after verifying the message record formats used by the translation.
- 4 Use a test site to check out translations before incorporating them into a production system.
- 5 When integration testing is completed, perform the final system test by running the engine and moving messages through it.

Test output

As a test is configured, a command-line command and requisite parameters are generated in **Preview Command to Issue**. This is for display only. A command cannot be manually specified. Click **Run Command** to run the test. The results are shown in the Result pane.

Using Grep

The Testing Tool can display much information. When there is much output, it can be difficult to locate the cause of errors or a particular field.

The Grep function is a way to filter the output from the testing tools.

Select **Grep** to enable this feature. (By default, this is disabled.) Specify grep content in the text field. The search expression can be a search string or a wildcard. Selecting the check box and having no content results in an error message when **Run Command** is clicked.

Grep keeps recently entered content history for all of the testing tools. This history is removed when the Testing Tool is closed.

When **Grep** is selected, a `-w search expression` command parameter is appended to the command line. For example:

```
hclhl7test -w "ADT_*" -d 2 -a test.dat
```

The Testing Tool command line filters output going to StdOut to show only those lines containing the search string or matching the wildcard. Output directed to StdErr continues to go to StdErr and is not affected.

Grep supported special characters

- [(single bracket)
- & (ampersand)
- " (two single quotes)
- Chinese punctuation mark
- Chinese dash
- Chinese ellipsis

Searching the results

Use the **Search** text field to search the results for a specific string or wildcard.

To begin a search, specify a string and click **Search**. The search expression can be a search string or a wildcard.

The first matching string is highlighted. Clicking **Search** again searches for and highlights the next matching string.

If no match is found, then `No lines found to match search expression` displays in the results.

Output options

Select **Options > Erase output on each run** to remove the previous test's output before running a test.

Select **Options > Auto-scroll to top of the output** to automatically show the test results before the last test's output. When this option is disabled, each test's output is added to the bottom of the list. This option is unavailable when **Erase output on each run** is selected.

Testing the database protocol

This calls the engine command line to test the specified database protocol thread. Under advanced security mode, the security server controls access to the command line.

- 1 Open the **Select Source Thread** menu to select the source thread. This lists all database protocol threads under the current site. If the selected thread is a Database Inbound protocol, then the Outbound Options group is disabled. The command line also runs the thread read action and converts the result to the VRL data.
- 2 If required, then select **Update Database** to update, insert, and delete data in the database through the generated command.
- 3 Specify the testing data file in **Choose Data File**, or click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITE` folder.

If the selected thread is a Database Outbound protocol, then specify the name of the input data file here. The format of the testing data file must conform to the VRL file of the database table schema defined in the Database Outbound protocol. Generally, it is a text file with each row corresponding to one row in the database table. The columns are separated by a comma.

To maintain security, browsing is restricted to within the `$HCIRoot` hierarchy.

- 4 Select how to process records.
process all records reads the selected data file and processes all the records in it.
process one record reads the selected data file and processes only the first record in the file.
- 5 Open the **Detail level** menu to select from a list of detail levels for the test output that is reported in the Result pane. Detail levels go from 0 (raw, unparsed data) to 4 (most detail). The default detail level is 1.
- 6 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.
NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.
Length Encoded reads the first 10 characters to determine the length of the first message. It reads that many characters into a message, and sends it to the parser.
EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.
- 7 Specify the encoding in the **Encoding** field, or click the arrow to open a list of different encodings. These identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-e encoding` option is added to the corresponding command line.
- 8 Specify a file name in which to store the result in the **Save To** field, or click the folder button to open a file browser.
 If the selected thread is a Database Outbound protocol, then based on the thread outbound SQL statement, the command line reads the testing data file. It then generates a list of SQL statements with filled values.
- 9 If required, then select **Grep** to filter the output.
- 10 Click **Run Command** to run the command shown in Preview Command to Issue.
 As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing the database schema

The database schema test parses the database schema format.

- 1 Open the **Connection** menu to select a database connection from the list.
- 2 Open the **Table Schema** menu to select a schema from the list of currently imported schemas under the database connection.
- 3 Specify the testing data file in **Choose Data File**, or click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITE` folder.
- 4 Select how to process records.
process all records reads the selected data file and processes all the records in it.
process one record reads the selected data file and processes only the first record in the file.
- 5 Open the **Detail level** menu to select from a list of detail levels for the test output that is reported in the Result pane. Detail levels go from 0 (raw, unparsed data) to 4 (most detail).

- 6 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats:

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine the first message length. Then it reads that many characters into a message, and sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.
- 7 Specify the encoding in the **Encoding** field, or click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-e encoding` option is added to the corresponding command line.
- 8 If required, then select **Grep** to filter the output.
- 9 Click **Run Command** to run the command shown in Preview Command to Issue.

As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing DICOM configurations

This tests the parsing of test data against a specified DICOM, or LDL, layout.

DICOM (Digital Imaging and Communications in Medicine), is a standard communication mechanism when integrating various medical products in a hospital environment. The medical products that are involved include Modalities (CT, MR, X-Ray, and others), Workstations, Archives, Printers, and HIS/RIS devices.

The intention of DICOM is to define the communication capabilities of each product type. This permits products, supplied by different vendors, to be connected together to form an open, integrated diagnostic and treatment capability.

The DICOM test runs a DICOM ECHO test, which is a DICOM ping action on the DICOM remote SCP (Service Class Provider). This verifies the DICOM network between the system DICOM SCU (Service Class User) and the remote DICOM SCP.

- 1 **Peer Host** is the host name of the DICOM peer. Specify the host in the field, or accept the default of *hostname*. This is required.
- 2 **Peer Port** is the TCP/IP port number of the peer. Specify the port in the field, or accept the default of 11112. This is required.
- 3 **Calling AE Title** is the AE (Application Entity) title. Specify the calling title in the field, or accept the default of CL-ECHO-SCU.
- 4 **Called AE Title** is the AE title of the peer. Specify the called title in the field, or accept the default of ANY-SCP.
- 5 **Timeout (seconds)** is the time-out for DIMSE (DICOM Message Service Element) messaging. Specify the time-out in seconds, or accept the default of 60.

- 6 Open the **Log Level** to open a menu of log levels to run.
- 7 Click **Run Command** to run the DICOM ECHO command shown in Preview Command to Issue.

Testing EDIFACT configurations

This tests UN/EDIFACT record format configurations. This test reads, parses, and displays messages, ensuring that configurations match the input data.

- 1 For **EDIFACT Version**, click the arrow to open a list of available versions.
- 2 For **Variant**, click the arrow to open a list of available variants.
- 3 Specify the testing data file in **Choose Data File**. You can also click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITEDIR/formats/` folder.
- 4 Select **Show Field Names** to show field names.
- 5 Open the **Detail level** menu to select from a list of detail levels for the test output that is reported in the Result pane. Detail levels go from 0 (raw, unparsed data) to 4 (most detail).
- 6 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine the first message length. Then it reads that many characters into a message, and sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.

- 7 Specify the encoding in the **Encoding** field. You can also click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-e encoding` option is added to the corresponding command line.
- 8 Select how to process records.
 - process all records** reads the selected data file and processes all the records in it.
 - process one record** reads the selected data file and processes only the first record in the file.
- 9 Select **Test Parser** (default) or **Test Numbers**.
 - Test Parser** parses each message.
 - Test Numbers** tests the number conversion.
- 10 In Separator Options, specify the separator characters for parsing messages.
See [Encoding separators](#).
- 11 If required, then select **Grep** to filter the output.
- 12 Click **Run Command** to run the command shown in Preview Command to Issue.

As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing FRL configurations

This tests fixed-record layout configurations. This test reads, parses, and displays messages, ensuring that configurations match the input data.

- 1 Specify an FRL file in the **FRL File** field. You can also click the folder button to open a file browser from which to select the file to test.
- 2 Specify the testing data file in **Choose Data File**, or click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITEDIR/formats/` folder.
- 3 Open the **Detail level** menu to select from a list of detail levels for the test output that is reported in the Result pane. Detail levels go from 0 (raw, unparsed data) to 4 (most detail).
- 4 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine the first message length. Then it reads that many characters into a message, and sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.

- 5 Specify the encoding in the **Encoding** field. You can also click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-e encoding` option is added to the corresponding command line.
- 6 Select how to process records.

process all records reads the selected data file and processes all the records in it.

process one record reads the selected data file and processes only the first record in the file.
- 7 If required, then select **Grep** to filter the output.
- 8 Click **Run Command** to run the command shown in Preview Command to Issue.

As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing HL7 configurations

This tests HL7 record format configurations. This test reads, parses, and displays messages, ensuring that configurations match the input data.

- 1 For **HL7 Version**, click the arrow to open a list of available versions.
- 2 For **Variant**, click the arrow to open a list of available variants.
- 3 Specify the testing data file in **Choose Data File**. You can also click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITEDIR/formats/` folder.
- 4 Select **Show Field Names** to show field names.

- 5 Open the **Detail level** menu to select from a list of detail levels for the test output that is reported in the Result pane. Detail levels go from 0 (raw, unparsed data) to 4 (most detail).
- 6 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine the first message length. Then it reads that many characters into a message, and sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.
- 7 Specify the encoding in the **Encoding** field. You can also click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-e encoding` option is added to the corresponding command line.
- 8 Select how to process records.

process all records reads the selected data file and processes all the records in it.

process one record reads the selected data file and processes only the first record in the file.
- 9 If required, then select **Grep** to filter the output.
- 10 Click **Run Command** to run the command shown in Preview Command to Issue.

As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing HPRIM configurations

This tests HPRIM record format configurations. This test reads, parses, and displays messages, ensuring that configurations match the input data.

- 1 For **HPRIM Version**, click the arrow to open a list of available versions.
- 2 For **Variant**, click the arrow to open a list of available variants.
- 3 Specify the testing data file in **Choose Data File**. You can also click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITEDIR/formats/` folder.
- 4 Select **Show Field Names** to show field names.
- 5 Open the **Detail level** menu to select from a list of detail levels for the test output that is reported in the Result pane. Detail levels go from 0 (raw, unparsed data) to 4 (most detail).
- 6 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine the first message length. Then it reads that many characters into a message, and sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.

- 7 Specify the encoding in the **Encoding** field. You can also click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-e encoding` option is added to the corresponding command line.
- 8 Select how to process records.

process all records reads the selected data file and processes all the records in it.

process one record reads the selected data file and processes only the first record in the file.
- 9 If required, then select **Grep** to filter the output.
- 10 Click **Run Command** to run the command shown in Preview Command to Issue.

As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing HRL configurations

This tests hierarchical-record layout configurations. This test reads, parses, and displays messages, ensuring that configurations match the input data.

- 1 Specify an FRL file in the **HRL File** field, or click the folder button to open a file browser from which to select the file to test.
- 2 Specify the testing data file in **Choose Data File**. You can also click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITEDIR/formats/` folder.
- 3 Open the **Detail level** menu to select from a list of detail levels for the test output that is reported in the Result pane. Detail levels go from 0 (raw, unparsed data) to 4 (most detail).
- 4 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine the first message length. Then it reads that many characters into a message, and sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.
- 5 Specify the encoding in the **Encoding** field. You can also click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-e encoding` option is added to the corresponding command line.
- 6 Select how to process records.

process all records reads the selected data file and processes all the records in it.

process one record reads the selected data file and processes only the first record in the file.

- 7 If required, then select **Grep** to filter the output.
- 8 Click **Run Command** to run the command shown in Preview Command to Issue.
As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing JSON configurations

The JSON data message file is based on the specified JSON schema file. This tests the JSON data message file that is based on the specified JSON schema file. You can select from the current site, master site, and root-level JSON schema files.

- 1 For **Package** open the menu and select the schema data file.
- 2 For **JSON Def** open the menu and select the JSON schema configuration file.
- 3 Specify the testing data file in **Choose Data File**. You can also click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITEDIR/formats/` folder.
- 4 Open the **Detail level** menu to select from a list of detail levels for the test output that is reported in the Result pane. Detail levels go from 0 (raw, unparsed data) to 4 (most detail).
- 5 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine the first message length. Then it reads that many characters into a message, and sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.

- 6 Specify the encoding in the **Encoding** field. You can also click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-e encoding` option is added to the corresponding command line.
- 7 Select how to process records
process all records reads the selected data file and processes all the records in it.
process one record reads the selected data file and processes only the first record in the file.
- 8 If required, then select **Grep** to filter the output.
- 9 Click **Run Command** to run the command shown in Preview Command to Issue.
As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing LDL configurations

LDL (Length Delimited Layout) is a data layout that is used for DICOM. The functionality is similar to FRL and VRL.

- 1 For **LDL Version**, click the arrow to open a list of available versions.
- 2 For **Variant**, click the arrow to open a list of available variants.
- 3 For **Messages**, click the arrow to open a list of messages defined in the variant. A message is composed of segments, such as HL7 messages. The standard LDL format does not include messages, although a variant of LDL should define some messages.
- 4 For **Transfer Syntax**, click the arrow to select the transfer syntax for the DICOM message. This is the encoding used for the exchange of DICOM information objects and messages, for example, Implicit VR-Little Endian; UID 1.2.840.10008.1.2. Application entities can unambiguously negotiate the encoding techniques they support. By this, the application entities can communicate. For example, data element structure, byte ordering, and compression.
- 5 Specify the testing data file in **Choose Data File**. You can also click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITEDIR/formats/` folder.
- 6 Select **Show Field Names** to show field names.
- 7 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine the first message length. Then it reads that many characters into a message, and sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.

- 8 Specify the encoding in the **Encoding** field. You can also click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-e encoding` option is added to the corresponding command line.
- 9 Select how to process records.
 - process all records** reads the selected data file and processes all the records in it.
 - process one record** reads the selected data file and processes only the first record in the file.
- 10 If required, then select **Grep** to filter the output.
- 11 Click **Run Command** to run the command shown in Preview Command to Issue.

As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing routes

Route specifications are the paths where messages flow from source threads to destination threads. Construct route specifications using a "From the Connection" approach.

Use the **Routes** tab to simulate the translation thread. Given a message, it determines the transaction ID and brings up the appropriate route configuration. This tool gets most of its configuration information from the Network Configurator.

The Route Testing Tool supports testing route configurations that contain DICOM transactions.

`hciroutetest` runs any inbound TPS for the source thread before it passes the messages to route and translate. This is necessary because the TPS often sets up metadata that is required for further processing.

If possible, then configure and verify complex routes without TPS modules. As the route is verified, add the modules in a logical order. The systematic insertion of TPS modules into the route specification makes isolating potential problems quicker.

It is important to verify each route. Valid message record formats for a thread and valid translations do not guarantee that the messages arrive at their correct destinations.

Test the route specifications for a particular source thread before starting the engine, after all message record formats and translations that are used by that thread are verified.

For `hciroutetest` to run translate procs in startup mode, select the **Run Translation Procs in Start Mode** check box on the **Process Configuration** dialog box. This is found at **Process > Configure** in Network Configurator.

Note: Global variable support is available for the `hciroutetest` command line tool. The global variable indicator `$$` (double dollar) has a conflict with the shell special variable, which has meaning as the current PID on Linux/AIX platforms. To use a global variable in the Testing Tool on Linux/AIX platforms, escape the `$$` using one of these forms. For example:

- `hciroutetest -a '$$variablename'`
- `hciroutetest -a \\\$variablename`

On Windows, use `hciroutetest -a $$variablename`.

- 1 For **Select Source Thread**, click the arrow to open a list of available threads.
- 2 Specify the testing data file in **Choose Data File**. You can also click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITEDIR/formats/` folder.
- 3 Select **Leak Detection** to check for memory leaks.
- 4 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine how long the first message is, reads that many characters into a message, and then sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.

- 5 Specify the encoding in the **Encoding** field, or click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from

the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-x` encoding option is added to the corresponding command line.

- 6 Select how to process records.

process all records reads the selected data file and processes all the records in it.

process one record reads the selected data file and processes only the first record in the file.

- 7 Select **Inbound Type Data** to test the data routes configured in the Network Configurator.
- 8 Select **Inbound Type Reply** and select a **Reply Thread** to test the reply routes configured in the Network Configurator. This is available only when **Inbound Type Reply** is selected.
- 9 Select **OB TPS** for all outbound TPS that are defined in the route path destination threads to run once with start mode. If both OB TPS and Send to Proc are selected, then the outbound TPS are run before the procedure is run. Output messages become input messages to the procedure. TPS uses the same disposition policy as the translation post queue (the message for KILL, OVER, and ERROR dispositions are discarded). OB TPS and Send to Proc use different Tcl interpreters. OB TPS uses the same Tcl interpreter as IB TPS.

If the check box is selected, then there must be a defined outbound TPS; otherwise, a warning results.

- 10 Select **Show Source Message ID** to show the input source message ID in the output.

This option displays the message between the translation and destination in `hciroutetest`.

When this option is selected and **Send to proc** is `hciroutetestshowbydest`, then the message between the translation and destination is displayed. Example:

```
test1.xlt msg
test2.xlt msg
dest msg
```

If the option is enabled and **Save to file** is enabled, then the message between the translation and destination is saved to file. Example:

```
Run hciroutetest there are files aaa, aaa.branch_dest1 and aaa.branch_dest2, messages after
test2.xlt and
test3.xlt saved in file aaa, messages after test3.xlt and the branch_dest1 msg saved in file
aaa.branch_dest1,
messages after test1.xlt and the branch_dest2 saved in file aa.branch_dest2.
```

For remote commands, this option is `-n`.

- 11 Select **Save To File** to save messages to a file. Specify the file name in the field. When this is selected, CONTINUE messages are saved to `base.dest.thread`. If `base` is not specified, then it defaults to `routeout`.
- 12 Select **Send To Proc** to control the message disposition and save to a Tcl end procedure. This enables the selection list. Click the arrow to select from a list of procs.
- 13 In the **Arguments** field, specify any arguments to pass in. This field is available when **Send To Proc** is selected.
- 14 Select **Error Database Proc** to enable the control for failed messages. Click **Edit** to open the TPS Editor. Then, click **Add** to open the **TPS Properties** dialog box, where you can select the procedure or Java class and arguments for failed messages.
Only one procedure or Java Class is supported.
When the check box is cleared, the error TPS that is defined in NetConfig is used.
- 15 If required, then select **Grep** to filter the output.
- 16 Click **Run Command** to run the command shown in Preview Command to Issue.

As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing TPS

This simulates most TPS UPoCs. TPS is a series of Tcl procedures composed of procedure argument elements, where:

- **Procs** lists the name of the procedure.
- **Arguments** lists any special arguments to pass in through the ARGS key.

Tcl Procedure Stream (TPS) modules permit the alteration of message processing by setting each message's disposition before exiting the module. In addition, these entry points let you modify a message's data, metadata, or flow.

Use Script Editor to build any TPS.

If specifying a file location for test output, then use the "/" as the file delimiter (all platforms) or begin the file path with a "\\" (Windows).

Use `hcitptest` to verify modules in a TPS. Test each module in the stream with `hcitptest` before placing the next module into the stream.

This process might require the use of an end-processing module to verify that the messages are leaving the module. It can also verify that a module stream has the correct disposition.

It is important to verify each TPS module before configuring it into your system. Although TPS provides substantial capability to modify data, an erroneous application of TPS functions can result in messages being delivered incorrectly, or not being delivered at all.

Additionally, you should verify each module in the TPS. The return value of each procedure affects the arguments of the subsequent procedure, and, depending on the procedure's return value, a subsequent procedure might not be invoked.

TPS can run in run, startup, time, or shutdown mode, although certain protocol thread Tcl procedures are only run in Startup mode during thread start-up.

The system sets the Tcl system encoding as UTF-8, so the default file encoding for input/output is in UTF-8.

The default encoding of `StdIn`, `StdOut`, and `StdErr` is ANSI. Use the `fconfigure` command to change the encoding of `StdIn`, `StdOut`, and `StdErr`. For example, to change the encoding to UTF-8 when you use the TPS Test Tool in the IDE.

Message data that is obtained or changed by Tcl is always UTF-8. A warning is given to remind you that string lengths are in bytes and are not the same as character (code point) lengths.

Note: Global variable support is available for the `hcitptest` command line tool. The global variable indicator `$$` (double dollar) has a conflict with the shell special variable, which has meaning as the current PID on Linux/AIX platforms. To use a global variable in the Testing Tool on Linux/AIX platforms, escape the `$$` using one of these forms. For example:

- `hcitptest -a '$$variablename'`

- `hcitptest -a \\\$variablename`

On Windows, use `hcitptest -a $$variablename`.

- 1 Select the **Procs Edit** button to open the **TPS Editor** dialog box, where you can select the TPS procedures to run when the command is run.
- 2 Specify the testing data file in **Choose Data File**. You can also click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITEDIR/formats/` folder.
- 3 **Leak Detection**: Select to check for memory leaks.
- 4 **Mode**: Click the arrow to open a list of modes. TPS can run in Run, Startup, Time, or Shutdown mode, although certain protocol thread Tcl procedures are only run in Startup mode during thread start-up.
- 5 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine how long the first message is, reads that many characters into a message, and then sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.
- 6 Specify the encoding in the **Encoding** field, or click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-x` encoding option is added to the corresponding command line.
- 7 Select how to process records.

process all records reads the selected data file and processes all the records in it.

process one record reads the selected data file and processes only the first record in the file.
- 8 Click the **Caller Context** arrow to select from a list of caller context names.
- 9 Select **Initial Startup** to start the connection.
- 10 For **Interval**, specify the interval to wait between calls. This option is available when **Time** is chosen for the mode.
- 11 For **Max Messages**, specify the maximum number of messages. This option is available when **Time** is chosen for the mode.
- 12 Select **Save To File** to save the message to a file. These are saved to `base.KILL` or `base.CONTINUE`, depending on the disposition returned by the TPS. Specify the file name in the field.
- 13 Select **Send To Proc** to save to a Tcl end procedure. This enables the selection list. Click the arrow to select from a list of procedures. The default is `hcitptestshowbydisp`.
- 14 For **Arguments**, specify any arguments to pass in. This text box is available when **Send To Proc** is selected.
- 15 If required, then select **Grep** to filter the output.
- 16 Click **Run Command** to run the command shown in Preview Command to Issue.

As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing a transaction ID

Tests a transaction ID (TRX ID) separately from the engine. This simulates getting the transaction ID on an incoming message from the source thread.

This runs any inbound TPS for the source thread before the engine retrieves the transaction ID. This is necessary because the TPS may change the message content.

- 1 For **Select Source Thread**, click the arrow to open a list of available threads.
- 2 Specify the testing data file in **Choose Data File**. You can also click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITEDIR/formats/` folder.
- 3 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine how long the first message is, reads that many characters into a message, and then sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.

- 4 Specify the encoding in the **Encoding** field. You can also click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-x encoding` option is added to the corresponding command line.
- 5 For **Driver Control**, set the content to message metadata DRIVERCONTROL. You can input any content to set to drivercontrol using this option.

For tests on DICOM messages, `-c drivercontrol` is required, where *drivercontrol* is `{AbstractSyntax***}{TransferSyntax***}`. This is required to get the TRX ID.

See [DICOM abstract syntax values](#) and [DICOM transfer syntax values](#).

- 6 Select how to process records.
 - process all records** reads the selected data file and processes all the records in it.
 - process one record** reads the selected data file and processes only the first record in the file.
- 7 Select **Inbound Type Data** to test the data routes configured in the Network Configurator.
- 8 Select **Inbound Type Reply** and select a **Reply Thread** to test the reply routes configured in the Network Configurator. This is available only when **Inbound Type Reply** is selected.
- 9 Select **Verbose** to show verbose information. When this is checked, a `-v` option is added to the command line.
- 10 Click **Run Command** to run the command shown in Preview Command to Issue.

As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing VRL configurations

Tests variable-record layout configurations. This test reads, parses, and displays messages, ensuring that configurations match the input data.

- 1 Specify a VRL file in the **VRL File** field. You can also click the folder button to open a file browser from which to select the file to test.
- 2 Specify the testing data file in **Choose Data File**. You can also click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITEDIR/formats/` folder.
- 3 Open the **Detail level** menu to select from a list of detail levels for the test output that is reported in the Result pane. Detail levels go from 0 (raw, unparsed data) to 4 (most detail).
- 4 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine the first message length. Then it reads that many characters into a message, and sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.

- 5 Specify the encoding in the **Encoding** field. You can also click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-e encoding` option is added to the corresponding command line.
- 6 Select how to process records.
 - process all records** reads the selected data file and processes all the records in it.
 - process one record** reads the selected data file and processes only the first record in the file.
- 7 If required, then select **Grep** to filter the output.
- 8 Click **Run Command** to run the command shown in Preview Command to Issue.

As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing X12 configurations

Tests X12 record format configurations. This test reads, parses, and displays messages, ensuring that configurations match the input data.

`hcix12test` handles X12 messages that contain:

- Any number of interchanges (ISA..IEA).
- or
- A single group (GS..GE).

or

- A single transaction set (ST..SE).

hcix12test accepts interchanges, groups, or transaction sets.

If the message starts with ST or GS, then it is treated as an individual transaction set or an individual group, respectively. For ST and GS messages, use default separator characters (*, ~ and \).

If the message does not start with ST or GS, then it is treated as an interchange. hcix12test handles a message with any number of interchanges, each containing any number of groups, which can contain any number of transaction sets. For interchanges, the separator characters are extracted from the ISA segments.

If using group messages (GS to GE), then create a variant that defines the transaction sets in terms of groups and specify such a variant for the test. The path names for a group are different from the transaction set. This is because the GS becomes group zero and the first ST is 1(0).0(0).ST(0).

The group message variant uses this structure:

```
GS
{
    ST
    . . .
    SE
}
GE
```

- If an envelope is used, then it extracts the transaction sets and parses the individual transaction sets.
- If transaction sets are used, then hcix12test processes the message without extracting the transaction sets.

- 1 For **X12 Version**, click the arrow to open a list of available versions.
- 2 For **Variant**, click the arrow to open a list of available variants.
- 3 Specify the testing data file in **Choose Data File**. You can also click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITEDIR/formats/` folder.
- 4 Select **Show Field Names** to show the field names.
- 5 Open the **Detail level** menu to select from a list of detail levels for the test output that is reported in the Result pane. Detail levels go from 0 (raw, unparsed data) to 4 (most detail).
- 6 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine how long the first message is, reads that many characters into a message, and then sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.

- 7 Specify the encoding in the **Encoding** field. You can also click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-e encoding` option is added to the corresponding command line.
- 8 Select how to process records.

process all records reads the selected data file and processes all the records in it.

process one record reads the selected data file and processes only the first record in the file.

- 9 Select **Test Parser** (default) or **Test Numbers**.

Test Parser parses each message.

Test Numbers tests the number conversion.

- 10 In Separator Options, specify the separator characters for parsing messages.

See [Encoding separators](#).

- 11 If required, then select **Grep** to filter the output.

- 12 Click **Run Command** to run the command shown in Preview Command to Issue.

As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing NCPDP Telecom configurations

Tests NCPDP Telecom record format configurations. This test reads, parses, and displays messages, ensuring that configurations match the input data.

Note: NCPDP Telecom, NCPDP SCRIPT, and NCPDP Formulary and Benefit require an additional license to become active.

- 1 For **NCPDP Telecom Version**, click the arrow to open a list of available versions.
- 2 For **Variant**, click the arrow to open a list of available variants.
- 3 Specify the testing data file in **Choose Data File**. You can also click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITEDIR/formats/` folder.
- 4 Select **Show Field Names** to show field names.
- 5 Open the **Detail level** menu to select from a list of detail levels for the test output that is reported in the Result pane. Detail levels go from 0 (raw, unparsed data) to 4 (most detail).
- 6 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine the first message length. Then it reads that many characters into a message, and sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.

- 7 Specify the encoding in the **Encoding** field. You can also click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-e encoding` option is added to the corresponding command line.
- 8 Select how to process records.

process all records reads the selected data file and processes all the records in it.

process one record reads the selected data file and processes only the first record in the file.

- 9 Select **Test Parser** (default) or **Test Numbers**.

Test Parser parses each message.

Test Numbers tests the number conversion.

- 10 If required, then select **Grep** to filter the output.

- 11 Click **Run Command** to run the command shown in Preview Command to Issue.

As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing NCPDP SCRIPT configurations

Tests NCPDP SCRIPT record format configurations. This test reads, parses, and displays messages, ensuring that configurations match the input data.

Note: NCPDP Telecom, NCPDP SCRIPT, and NCPDP Formulary and Benefit require an additional license to become active.

- 1 For **NCPDP SCRIPT Version**, click the arrow to open a list of available versions.
- 2 For **Variant**, click the arrow to open a list of available variants.
- 3 Specify the testing data file in **Choose Data File**. You can also click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITEDIR/formats/` folder.
- 4 Select **Show Field Names** to show field names.
- 5 Open the **Detail level** menu to select from a list of detail levels for the test output that is reported in the Result pane. Detail levels go from 0 (raw, unparsed data) to 4 (most detail).
- 6 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine the first message length. Then it reads that many characters into a message, and sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.

- 7 Specify the encoding in the **Encoding** field. You can also click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-e encoding` option is added to the corresponding command line.
- 8 Select how to process records.

process all records reads the selected data file and processes all the records in it.

process one record reads the selected data file and processes only the first record in the file.

- 9 Select **Test Parser** (default) or **Test Numbers**.
Test Parser parses each message.
Test Numbers tests the number conversion.
- 10 In Separator Options, specify the separator characters for parsing messages.
 See [Encoding separators](#).
- 11 If required, then select **Grep** to filter the output.
- 12 Click **Run Command** to run the command shown in Preview Command to Issue.
 As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing NCPDP F&B configurations

Tests NCPDP Formulary and Benefit record format configurations. This test reads, parses, and displays messages, ensuring that configurations match the input data.

Note: NCPDP Telecom, NCPDP SCRIPT, and NCPDP Formulary and Benefit require an additional license to become active.

- 1 For **NCPDP F&B Version**, click the arrow to open a list of available versions.
- 2 For **Variant**, click the arrow to open a list of available variants.
- 3 Specify the testing data file in **Choose Data File**. You can also click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITEDIR/formats/` folder.
- 4 Select **Show Field Names** to show field names.
- 5 Open the **Detail level** menu to select from a list of detail levels for the test output that is reported in the Result pane. Detail levels go from 0 (raw, unparsed data) to 4 (most detail).
- 6 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine the first message length. Then it reads that many characters into a message, and sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.

- 7 Specify the encoding in the **Encoding** field. You can also click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-e encoding` option is added to the corresponding command line.
- 8 Select how to process records
process all records reads the selected data file and processes all the records in it.
process one record reads the selected data file and processes only the first record in the file.

- 9 Select **Test Parser** (default) or **Test Numbers**.

Test Parser parses each message.

Test Numbers tests the number conversion.

- 10 If required, then select **Grep** to filter the output.

- 11 Click **Run Command** to run the command shown in Preview Command to Issue.

As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

12

Testing XLT configurations

`hcixlttest` tests a translation specification without starting the engine. This tool requires only the name of the translation. This defines a translation specification that converts translation data into another format.

- If an input file is not supplied, then the tool checks only the translation specification for validity.
- If an input file is supplied, then the tool processes the messages in the input file.

You can configure a translation file to modify a DICOM data element value. DICOM can be used as input or output for a translation.

- 1 For **Translation File**, click the folder button to open a file browser from which to select the file to test.
- 2 Specify the testing data file in **Choose Data File**. You can also click the folder button to open a file browser to select the data file. By default, the file browser locates on the `$HCISITEDIR/formats/` folder.
- 3 Select how to process records.
process all records reads the selected data file and processes all the records in it.
process one record reads the selected data file and processes only the first record in the file.
- 4 Select **This defines a translation specification that converts translation data into** **Show Field Names** to show field names.
- 5 Select **Leak Detection** to check for memory leaks.
- 6 Select **Print unreadable chars without changing to hex** to print the output in pretty print.
- 7 Open the **Detail level** menu to select from a list of detail levels for the test output that is reported in the Result pane. Detail levels go from 0 (raw, unparsed data) to 4 (most detail).
- 8 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine the first message length. Then it reads that many characters into a message, and sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.

- 9 Specify the encoding in the **Encoding** field. You can also click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-e encoding` option is added to the corresponding command line.
- 10 For **Pre Xlate TPS**, click **Edit** to open the **TPS Editor** dialog box. Use this dialog box to select the TPS procedures to run before message translation. Click **Add** to select the procedures or Java class. Then, specify any arguments.
- 11 For **Post Xlate TPS**, click **Edit** to open the **TPS Editor** dialog box. Use this dialog box to select the TPS procedures to run after message translation. Click **Add** to select the procedures or Java class. Then, specify any arguments.
- 12 Specify a file name in which to store the result in the **Save To** field, or click the folder button to open a file browser.
- 13 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine the first message length. Then it reads that many characters into a message, and sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.
- 14 In Inbound Separator Options and Outbound Separator Options, specify the separator characters for parsing messages.
See [Encoding separators](#).
- 15 If required, then select **Grep** to filter the output.
- 16 Click **Run Command** to run the command shown in Preview Command to Issue.
As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing XML configurations

Parses an XML test message against a DTD/Schema/DTD-containing-XML file. The test process validates the consistency of the message and DTD/Schema/DTD-containing-XML.

You can also explicitly name the message definition to test.

Before running the XML test, place on the server a compatible pair of files consisting of a test XML message and a DTD/Schema/DTD-containing-XML file.

A pre-test is required on the DTD/Schema/DTD-containing-XML file. This pre-test is performed through the **Compile** option on the XML Package Manager.

After the pre-test is finished, take note of the package containing the pre-tested DTD/Schema/DTD-containing-XML file. This pre-tested file must be referenced within the XML test message that is used in the Testing Tool validation and test parse.

- 1 For **Package**, click the arrow to open a list of available packages on the server machine (HCIR00T\HCISITE\formats\xml\). Select the package containing the prepared DTD/Schema/DTD-containing-XML file.
- 2 For **XML**, select the message definition from the list.
- 3 Specify the testing data file in **Choose Data File**. You can also click the folder button to open a file browser to select the data file. By default, the file browser locates on the \$HCISITEDIR/formats/ folder.
- 4 Open the **Detail level** menu to select from a list of detail levels for the test output that is reported in the Result pane. Detail levels go from 0 (raw, unparsed data) to 4 (most detail).
- 5 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine the first message length. Then it reads that many characters into a message, and sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.

- 6 Specify the encoding in the **Encoding** field. You can also click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an `-e encoding` option is added to the corresponding command line.
- 7 Select how to process records.

process all records reads the selected data file and processes all the records in it.

process one record reads the selected data file and processes only the first record in the file.
- 8 Select **Print unreadable chars without changing to hex** to print the output in pretty print.
- 9 If required, then select **Grep** to filter the output.
- 10 Click **Run Command** to run the command shown in Preview Command to Issue.

As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Testing XSLT configurations

XSLT provides the ability to use parameters specified at runtime through the use of the element `<xsl:param/>`. To pass runtime parameters to the XSLT processor, you can set it inside USERDATA message meta-data. This is under the XSLTRuntimeParams key in the Pre-Proc of the XSLT Route Details for XSLT.

For example:

```
keylget args MSGID mh
# get old USERDATA value
```

```
set userData [msgmetaget $mh USERDATA]
# set runtime parameters
keylset userData XSLTRuntimeParams {{PARAM1 Value1} {PARAM2 Value2}}
msgmetaset $mh USERDATA $userData
lappend dispList "CONTINUE $mh"
```

- XSLTRuntimeParams contains a keyed list of parameter-value pairs, for example, {{PARAM1 Value1} {PARAM2 Value2}}.
- The **Runtime Parameters** field can take a Tcl keyed list of parameter-value pairs that are embraced by double quotes as input.
- The value entry of each pair should contain a valid XPath expression. In particular, string literals should be enclosed in single quotes. Special characters in Tcl should be escaped using the forward slash.

For example, this is a list of valid runtime parameters:

```
"{NAME {'John Doe'}} {COMMENTS {'THIS IS A \{{test\}}'} {XPATH_VALUE {PEOPLE/PERSON}}"
```

- 1 Specify an XSLT file in the **XSLT File** field. You can also click the folder button to open a file browser from which to select the file to test.
- 2 Specify the testing data file in **Choose Data File**. You can also click the folder button to open a file browser to select the data file. By default, the file browser locates on the \$HCISITEDIR/formats/ folder.
- 3 Specify **Runtime Parameters** if required. This field is optional. This takes a Tcl keyed list of parameter-value pairs as input.
- 4 Specify the encoding in the **Encoding** field. You can also click the arrow to open a list of different encodings that identify the encoding of messages in the selected data file. The command converts the messages from the identified encoding to UTF-8 to perform the test. When you select an encoding from the list, an -e encoding option is added to the corresponding command line.
- 5 Specify a file name in which to store the output message in the **Save To** field. You can also click the folder button to open a file browser.
- 6 Select the line termination format in which to save the test messages. Click the arrow to open a list of formats.

NewLine Terminated reads the data in the file until it finds a newline character, making all that data one message, and sends that to the parser. It then reads until it finds the next newline character, makes a second message, and sends that to the parser.

Length Encoded reads the first 10 characters to determine the first message length. Then it reads that many characters into a message, and sends it to the parser.

EOF Terminated reads the file until it gets to the end-of-file character, takes that as a message, and sends it to the parser.
- 7 If required, then select **Grep** to filter the output.
- 8 Click **Run Command** to run the command shown in Preview Command to Issue.

As the test is being configured, a command-line command and the requisite parameters are generated in Preview Command to Issue. This is for display only. A command cannot be manually entered.

Using XSLT files to validate

The system supports XSLT validation files for HL7 CDA 2.0 and HITSP_C32 (2.5 version). These files are located in the root level XSLT folder (\$HCIRoot/xslt).

They can be used by `hcixslttest` or the engine without any changes. It can also be copied and modified to the site level for special use. This is at \$HCISITEDIR/xslt or \$HCIMASTERSITEDIR/xslt.

You should have a document file to be validated according to a particular specification. Generally, this is an XML file, for example, `document.xml`.

- 1 Prepare the validation XSLT files. Because Cloverleaf provides XSLT validation files for standard HL7 CDA R2 and HITSP C32 version 2.5, the engine and GUI can support root level XSLT files. You can directly use these XSLT files under the root level or move them to the site level and modify them for a particular purpose.
- 2 Use `hcixslttest` to do the validation from the GUI or command line.

From the GUI:

- a Launch the Testing Tool in the GUI.
- b Select the **XSLT** tab.
- c Set the validation files in **XSLT File**. For example, `ccd_iso_pretty.xsl` for HL7 CDA R2.
- d In the **Choose Date File** browser, set the file which needs validation. For example, `document.xml`.
- e Set any other options as required.
- f Click **Run Command** and get the validation report in the Result pane.

From the command line:

- a Run this command:

```
hcixslttest [-e <encoding>] [-F <format>] [-p <runtime_parameters>]
[-w searchExp] <xslt> <infile> [<outfile>]
```

For example:

```
hcixslttest -e ASCII CDA_2.0/ccd_iso_pretty.xsl document.xml
```

- b View the results.

The test result shows the validation report, indicating any errors or warnings where the `document.xml` file violates the standard specifications.

XSLT examples

Example: Valid document.xml

`hcixslttest` shows all check phases on the `document.xml` without any warning/error information:

```
MESSAGE 1
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<svrl:schematron-output xmlns:svrl="http://purl.oclc.org/dsdl/svrl">
```

```

xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:schold="http://www.ascc.net/xml/schematron"
xmlns:iso="http://purl.oclc.org/dsdl/schematron" xmlns:cda="urn:hl7-org:v3"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" title="Schematron schema for validating
conformance to CCD documents" schemaVersion="">
<svrl:ns-prefix-in-attribute-values uri="urn:hl7-org:v3" prefix="cda"/>
<svrl:ns-prefix-in-attribute-values uri="http://www.w3.org/2001/XMLSchema-instance" prefix="xsi"/>
<svrl:active-pattern id="p-2.16.840.1.113883.10.20.1-errors" name="CCD v1.0 Templates Root - errors
validation phase"/>
<svrl:active-pattern id="p-2.16.840.1.113883.10.20.1-warning" name="CCD v1.0 Templates Root -
warning validation phase"/>
<svrl:active-pattern id="p-2.16.840.1.113883.10.20.1-manual" name="CCD v1.0 Templates Root -
manual validation phase"/>
...
<svrl:active-pattern id="p-2.16.840.1.113883.10.20.1.58-warning" name="Verification of an advance
directive observation - warning validation phase"/>
<svrl:active-pattern id="p-2.16.840.1.113883.10.20.1.58-manual" name="Verification of an advance
directive observation - manual validation phase"/>
</svrl:schematron-output>

```

Example: Invalid document.xml

hcixslttest shows all check phases on the document.xml and indicates warning/error information:

- Note:

```

<svrl:failed-assert test="cda:providerOrganization" location="/*:ClinicalDocument[namespace-
uri()='urn:hl7-org:v3']
[1]/*:recordTarget[namespace-uri()='urn:hl7-org:v3'] [1]/*:patientRole[namespace-uri()='urn:hl7-
org:v3'] [1]">
<svrl:text>

```

- Note:

```

The providerOrganization element MAY be present. See HL7 History and Physical Note, CONF-
HP-36.
</svrl:text>
</svrl:failed-assert>

```

- Warning:

```

<svrl:failed-assert test="//cda:templateId/@root=&quot;2.16.840.1.113883.3.88.11.83.5&quot;"
location="/*:ClinicalDocument[namespace-uri()='urn:hl7-org:v3'] [1]/*:component[namespace-
uri()='urn:hl7-org:v3']
[1]/*:structuredBody[namespace-uri()='urn:hl7-org:v3'] [1]/*:component[namespace-uri()='urn:hl7-
org:v3']
[1]/*:section[namespace-uri()='urn:hl7-org:v3'] [1]">
<svrl:text>

```

- Warning:

```

HITSP/C83 Clinical Document, the Payers section will include entries from the Insurance
Provider module when
this information is known. See HITSP/C83 Section 2.2.1.1, rule C83-[CT-101-2].
</svrl:text>
</svrl:failed-assert>

```

- Error:

```

<svrl:failed-assert test="//cda:templateId/@root=&quot;2.16.840.1.113883.3.88.11.83.14&quot;"

```

```
location="/*:ClinicalDocument[namespace-uri()='urn:hl7-org:v3'][1]/*:component[namespace-
uri()='urn:hl7-org:v3']
[1]/*:structuredBody[namespace-uri()='urn:hl7-org:v3'][1]/*:component[namespace-uri()='urn:hl7-
org:v3']
[8]/*:section[namespace-uri()='urn:hl7-org:v3'][1]">
<svrl:text>
```

- Error:

```
HITSP/C83 Clinical Document, the Vital Signs section SHALL contain entries conforming to the
Vital Sign module.
See HITSP/C83 Section 2.2.1.19, rule C83-[CT-119-2].
</svrl:text>
</svrl:failed-assert>
```

You can use the **Grep** option (`-w`) to search these keywords in the validation report.

TCP/IP

This configures, tests, and debugs TCP/IP connections.

`hcitcptest` is run from the command line and helps debug TCP/IP interfaces.

In Windows, **Start > Programs > Cloverleaf Integration Services 6.x > Tools > TCPIP** test starts the TCP/IP Test stand-alone GUI.

An external interface is a system or device that produces information for translation by the system or receives information produced by a system translation. Essentially, it is any physical device that is connected to the system hub. In terms of the system, there is one protocol thread per external interface.

Use the Network Configurator to build external interfaces.

The connection from the system to an external interface or device is a basic concept. The process of establishing connectivity and verifying the protocol is a challenging aspect of the network.

The primary reason for simulating the external interfaces is that the physical device might not be accessible for testing purposes. It also isolates any protocol issues.

In an effort to better isolate potential problems when dealing with external interfaces or devices, use a step-wise approach to verifying the connection protocol. The tool for verifying external interfaces is `hcitcptest`, which permits a manual interface to a connection.

Verifying external interfaces

These procedures are suggested when attempting to connect and verify an external interface to the system hub. Perform the first five steps before or after constructing any message record formats, translations, or TPS modules. You should perform the final step only after verifying all record formats, translations, TPS modules, and routes.

- 1 Verify physical connectivity to the external interface.
- 2 Verify that the connection is established. You should verify that communications to the device are valid in the required direction. For example, uni-directional versus bi-directional.
The goal of verifying physical connectivity is to ensure that the system can communicate with the external device in a basic sense. In other words, you can see valid data at each end point of the connection, if appropriate.
There are many tools available to aid in verifying physical connectivity. The tool to use depends on the specific protocol you are using. For example, you can use the UNIX `tip` utility to verify that an asynchronous connection works properly.
You can also use `hctcpctest` to test a TCP/IP connection.
- 3 Verify that the protocol to the external interface operates as specified by the vendor.
- 4 Verify that message formats are correct and that positive and negative acknowledgments work as specified.
- 5 Verify protocol characteristics, such as message resend and link reconnection strategies.
- 6 Verify that the characteristics of the protocol driver match those of the external interface. Use this step to configure and verify protocol-specific TPS modules.
- 7 Verify that the protocol can correctly communicate with the external interface. For this verification, configure a route with two protocol threads. Use the File protocol for one of the threads, and the protocol to test for the other thread. Messages can flow from a file to the external device or the external device to a file.
Note: All simulators must be removed from the network and the external interface that is connected to the system hub.

Testing TCP/IP connections

This configures, tests, and debugs TCP/IP connections.

- 1 Access the TCP/IP Test GUI.
From Windows, select **Start > Programs > Infor Cloverleaf Integration Suite > Tools > TCPIP Test**.
From the command line, specify `hciguitcpctest`.
- 2 Click **Options** or select **File > Options** to open the **Connection Options** dialog box.
Select **Log to File** to configure the log file in which to record the conversation and to specify whether to send line feed. Specify a file name or click **Browse** for the file name.
Select **Send line feed** to append a line feed at the end of the message.
Select the required **Message Encoding Type**. This determines how TCP/IP Test is encoded in the protocol.
The character encoding encodes the output string with the correct character encoding before it is sent out. It also decodes the input bytes with the correct character encoding before it is shown on the screen. The default is UTF-8.
- 3 Click **Apply** and **OK** to return to the main GUI.
- 4 In the main GUI, **Send Data** is where you edit the data to be sent to the remote application.
Received Data contains the data received from the remote application. No edits are possible.

5 Configure the server or client by clicking the tab.

Click the **Server** tab to specify a port on the local machine with which this utility listens. Click **Options** to open the **Connection Options** dialog box. Then, select the log file in which to record the conversation and to specify whether to send line feed.

Click the **Client** tab to specify a host and port with which to connect. Data received from the server program can optionally be written to a local file. Clicking **Connect** forces the socket connection.

HIPAA external code sets

External code sets are tables of acceptable code values and their associated descriptive information used in HIPAA standard transactions. They are referred to as external because they are defined by organizations other than the ones that define the transaction formats.

Code sets are viewed and modified with the Lookup Table Configurator.

HIPAA guide number 5: Countries, currencies, and funds

Table	Description
<code>countries.tbl</code>	Two-letter country code versus country name
<code>currencies.tbl</code>	Three-letter currency code versus currency name
<code>currenciesNum.tbl</code>	Three-letter currency code versus 3-number currency code

HIPAA guide number 22: States and outlying areas of the U.S.

Table	Description
<code>states.tbl</code>	Two-letter state/outlying area code versus name
<code>canProvinces.tbl</code>	Two-letter Canadian Province code versus name

HIPAA guide number 130: Health care financing administration common procedural coding system

Table	Description
hcpcsProcLong.tbl	Character procedure code versus long description
hcpcsProcShort.tbl	Five-character procedure code versus short description
hcpcsModLong.tbl	Two-letter modifier code versus long description
hcpcsModShort.tbl	Two-letter modifier code versus short description
hcpcsPriceInd.tbl	Procedure/modifier code versus pricing indicator code
hcpcsMultPriceInd.tbl	Procedure/modifier code versus multiple pricing ind. code
hcpcsCovIssues.tbl	Procedure/modifier code versus coverage issues
hcpcsMediCarMRS.tbl	Procedure/modifier code versus Medicare carriers manual reference section
hcpcsStatuteNum.tbl	Procedure/modifier code versus statute number
cpcsLabCertCode.tbl	Procedure/modifier code versus lab certification code
hcpcsCrossRefCode.tbl	Procedure/modifier code versus cross reference code
hcpcsCovCode.tbl	Procedure/modifier code versus coverage code
hcpcsAPayGrpCode.tbl	Procedure/modifier code versus ASC payment group code
hcpcsAPayGrpEffDate.tbl	Procedure/modifier code versus ASC payment group effective date
hcpcsMPayGrpCode.tbl (no entries)	Procedure/modifier code versus MOG payment group code
cpcsMPayPolInd.tbl (no entries)	Procedure/modifier code versus MOG payment policy Indicator
hcpcsMEffDate.tbl (no entries)	Procedure/modifier code versus MOG effective date
hcpcsProNoteNum.tbl	Procedure/modifier code versus processing note number
hcpcsBETSC.tbl	Procedure/modifier code versus Berenson-Eggers type of service code
hcpcsTSC.tbl	Procedure/modifier code versus type of service code

Table	Description
hcpcsAnesBUQ.tbl	Procedure/modifier code versus anesthesia base unit quantity
hcpcsCodeAddDate.tbl	Procedure/modifier code versus code added date
hcpcsActEffDate.tbl	Procedure/modifier code versus action effective date
hcpcsTermDate.tbl	Procedure/modifier code versus termination date
hcpcsActionCode.tbl	Procedure/modifier code versus action code

HIPAA guide number 131: International classification of diseases clinical mod (ICD-9-CM) procedure

Table	Description
icdLong.tbl	Five-character code versus long description
icdMedium.tbl	Five-character code versus medium description
icdShort.tbl	Five-character code versus short description

HIPAA guide number 132: National uniform billing committee (NUBC) codes

To be provided in a future release

HIPAA guide number 133: Current procedural terminology (CPT) code

Table	Description
cptLong.tbl	Procedure code versus long description
cptMedium.tbl	Procedure code versus medium description
cptShort.tbl	Procedure code versus short description

Table	Description
cptModifiers.tbl	Modifier code versus long description

HIPAA guide number 134: National drug code

LISTING.TXT

Table	Description
ndcLabelCode.tbl	Numeric listing sequence number versus alpha label code
ndcProdCode.tbl	Numeric listing sequence number versus alpha product code
ndcStrength.tbl	Numeric listing sequence number versus alpha strength
ndcUnit.tbl	Numeric listing sequence number versus alpha unit
ndcRxOtc.tbl	Numeric listing sequence number versus 1-character Rx or OTC
ndcDosageForm.tbl	Numeric listing sequence number versus alpha dosage form
ndcFirmSeqNo.tbl	Numeric listing sequence number versus numeric firm seq num
ndcTradeName.tbl	Numeric listing sequence number versus product name

PACKAGES.TXT (duplicate values for listing sequence number)

Table	Description
ndcPackageCode.tbl	Listing sequence number concatenated with count for seq. num. versus package code
ndcPackageSize.tbl	Listing sequence number concatenated with count for seq. num. versus package size
ndcPackageType.tbl	Listing sequence number concatenated with count for seq. num. versus package type

FORMULAT.TXT (duplicate values for listing sequence number)

Table	Description
ndcFormulatStrength.tbl	Listing sequence number concatenated with count for seq. num. versus formulation strength
ndcFormulatUnit.tbl	Listing sequence number concatenated with count for seq. num. versus formulation unit
ndcFormulatIngName.tbl	Listing sequence number concatenated with count for seq. num. versus formulation ingredient name

DRUGCLAS.TXT (duplicate values for listing sequence number)

Table	Description
ndcDrugClassNum.tbl	Listing sequence number concatenated with count for seq. num. versus drug class number
ndcDrugClass.tbl	Listing sequence number concatenated with count for seq. num. versus drug class description

APPLICAT.TXT (duplicate values for listing sequence number)

Table	Description
ndcApplicationNumber.tbl	Listing sequence number concatenated with count for seq. num. versus application number
ndcAppProdNumber.tbl	Listing sequence number concatenated with count for seq. num. versus application product number

FIRMS.TXT

Table	Description
ndcFirmLabelCode.tbl	Numeric firm sequence number versus alpha label code
ndcFirmName.tbl	Numeric firm sequence number versus alpha firm name <compliance address: 9 fields>

ROUTES.TXT (duplicate values for listing sequence number)

Table	Description
ndcRouteCodes.tbl	Listing sequence number concatenated with count for seq. num. versus route of admin code

Table	Description
ndcRouteNames.tbl	Listing sequence number concatenated with count for seq. num. versus route name

TBLDCLAS.TXT

Table	Description
ndcTblDCLass.tbl	Numeric code versus drug class description

TBLDOSAG.TXT

Table	Description
ndcTblDosageLong.tbl	Numeric code versus dosage long description
ndcTblDosageShort.tbl	Numeric code versus dosage short description

TBLROUTE.TXT

Table	Description
ndcTblRoute.tbl	Numeric code versus route description

TBLUNIT.TXT

Table	Description
ndcTblUnit.tbl	Numeric code versus unit description

HIPAA guide number 135: American dental association code

Table	Description
adaLong.tbl	Code versus long description
adaShort.tbl	Code versus short description

HIPAA guide number 139: Claim adjustment reason code

Table	Description
carcDesc.tbl	Code versus description
carcNote.tbl	Code versus notes, if any

HIPAA guide number 235: Claim frequency type code

To be provided in a future release

HIPAA guide number 240: National drug code by format

Table	Description
ndcbfDosage.tbl	Numeric code versus dosage format description
ndcbfRoutes.tbl	Numeric code versus routes of administration description
ndcbfUnits.tbl	Alpha code versus unit description
ndcbfPackageType.tbl	Alpha code versus package type description

HIPAA guide number 245: National association of insurance commissioners (NAIC) code

Alien Insurers (alphabetic)

Table	Description
naicALAL_cn.tbl	Company code versus company name
naicALAL_cc.tbl	Company code versus company country

Alien Insurers (numeric)

Table	Description
naicALNM_cn.tbl	Company code versus company name
naicALNM_cc.tbl	Company code versus company country

Combined Property Companies

Table	Description
naicCOMB_cn.tbl	Company code versus company name
naicCOMB_gc.tbl	Company code versus group code
naicCOMB_cs.tbl	Company code versus company status

Companies (numeric)

Table	Description
naicCONM_gc.tbl	Company code versus group code
naicCONM_fn.tbl	Company code versus FEIN number
naicCONM_cs.tbl	Company code versus company status
naicCONM_sd.tbl	Company code versus state of domicile
naicCONM_cn.tbl	Company code versus company name

Fraternal Companies

Table	Description
naicFRAT_cn.tbl	Company code versus company name
naicFRAT_sd.tbl	Company code versus state of domicile
naicFRAT_gc.tbl	Company code versus group code
naicFRAT_fn.tbl	Company code versus FEIN number
naicFRAT_cs.tbl	Company code versus company status
naicFRAT_a.tbl	Company code versus address
naicFRAT_c.tbl	Company code versus city
naicFRAT_s.tbl	Company code versus state
naicFRAT_z.tbl	Company code versus zip
naicFRAT_p.tbl	Company code versus phone

Groups (alphabetic)

Table	Description
naicGPAL_gn.tbl	Group code versus group name

Groups (numeric)

Table	Description
naicGPNM_gn.tbl	Company code versus group name
naicGPNM_gc.tbl	Company code versus group code
naicGPNM_fn.tbl	Company code versus FEIN number
naicGPNM_cs.tbl	Company code versus company status
naicGPNM_sd.tbl	Company code versus state of domicile
naicGPNM_cn.tbl	Company code versus company name

HMDI Companies

Table	Description
naichMDI_cn.tbl	Company code versus company name
naichMDI_sd.tbl	Company code versus state of domicile
naichMDI_gc.tbl	Company code versus group code
naichMDI_fn.tbl	Company code versus FEIN number
naichMDI_cs.tbl	Company code versus company status
naichMDI_a.tbl	Company code versus address
naichMDI_c.tbl	Company code versus city
naichMDI_s.tbl	Company code versus state
naichMDI_z.tbl	Company code versus zip
naichMDI_p.tbl	Company code versus phone

HMO Companies

Table	Description
naichMO_cn.tbl	Company code versus company name
naichMO_sd.tbl	Company code versus state of domicile
naichMO_gc.tbl	Company code versus group code

Table	Description
naicHMO_fn.tbl	Company code versus FEIN number
naicHMO_cs.tbl	Company code versus company status
naicHMO_a.tbl	Company code versus address
naicHMO_c.tbl	Company code versus city
naicHMO_s.tbl	Company code versus state
naicHMO_z.tbl	Company code versus zip
naicHMO_p.tbl	Company code versus phone

LHSO Companies

Table	Description
naicLHSO_cn.tbl	Company code versus company name
naicLHSO_sd.tbl	Company code versus state of domicile
naicLHSO_gc.tbl	Company code versus group code
naicLHSO_fn.tbl	Company code versus FEIN number
naicLHSO_cs.tbl	Company code versus company status
naicLHSO_a.tbl	Company code versus address
naicLHSO_c.tbl	Company code versus city
naicLHSO_s.tbl	Company code versus state
naicLHSO_z.tbl	Company code versus zip
naicLHSO_p.tbl	Company code versus phone

Life Companies

Table	Description
naicLIFE_cn.tbl	Company code versus company name
naicLIFE_sd.tbl	Company code versus state of domicile
naicLIFE_gc.tbl	Company code versus group code
naicLIFE_fn.tbl	Company code versus FEIN number
naicLIFE_cs.tbl	Company code versus company status
naicLIFE_a.tbl	Company code versus address
naicLIFE_c.tbl	Company code versus city

Table	Description
naicLIFE_s.tbl	Company code versus state
naicLIFE_z.tbl	Company code versus zip
naicLIFE_p.tbl	Company code versus phone

Pools and Associations (alphabetic)

Table	Description
naicPLAL_pae.tbl	Pool number versus pool/association/entity
naicPLAL_sd.tbl	Pool number versus state of domicile

Pools and Associations (numeric)

Table	Description
naicPLNM_t.tbl	Pool number versus type
naicPLNM_pae.tbl	Pool number versus pool/association/entity
naicPLNM_sd.tbl	Pool number versus state of domicile

Property Companies

Table	Description
naicPROP_cn.tbl	Company code versus company name
naicPROP_sd.tbl	Company code versus state of domicile
naicPROP_gc.tbl	Company code versus group code
naicPROP_fn.tbl	Company code versus FEIN number
naicPROP_cs.tbl	Company code versus company status
naicPROP_a.tbl	Company code versus address
naicPROP_c.tbl	Company code versus city
naicPROP_s.tbl	Company code versus state
naicPROP_z.tbl	Company code versus zip
naicPROP_p.tbl	Company code versus phone

Title Companies

Table	Description
naicTILE_cn.tbl	Company code versus company name
naicTILE_sd.tbl	Company code versus state of domicile
naicTILE_gc.tbl	Company code versus group code
naicTILE_fn.tbl	Company code versus FEIN number
naicTILE_cs.tbl	Company code versus company status
naicTILE_a.tbl	Company code versus address
naicTILE_c.tbl	Company code versus city
naicTILE_s.tbl	Company code versus state
naicTILE_z.tbl	Company code versus zip
naicTILE_p.tbl	Company code versus phone
naicCompanyStatus.tbl	Company status code versus description

HIPAA guide number 507: Health care claim status category code

Table	Description
hccsccDesc.tbl	Code versus description
hccsccNote.tbl	Code versus notes, if any

HIPAA guide number 513: Home infusion EDI coalition (HIEC) product/service code list

Table	Description
hiecDesc.tbl	Code versus complete description
hiecSummDesc.tbl	Code versus summary description

HIPAA guide number 540: Health care financing administration national plan ID (subject to availability)

To be provided in a future release.

HIPAA guide number (no number): Provider taxonomy code

Table	Description
protaxTitles.tbl	Ten-character code versus title
protaxDescriptions.tbl	Ten-character code versus description

PDL

Programmable Driver Language (PDL) enables the development of drivers for specialized needs. The programmable driver module acts as a single driver programmable through PDL, a language specifically designed for writing drivers.

A PDL program usually consists of two main sections, which the user writes:

- The first section's phrases specify the structure of messages that are transmitted and received through the device. These phrases make up the vocabulary that the driver uses to communicate with its counterpart.
- The second section is plain Tcl code, which defines the entry points that handle protocol-level read, write, and other operations. The Tcl section can incorporate extensions and Tcl code for regular functionality.

Tcl enables the user to write the driver's program structure, and uses an existing language to define arbitrary protocols.

This section describes PDL version 1.2 and addresses the interface between the engine and the programmable driver module and the programmable driver module's internal structure.

How Tcl interacts with the protocol driver (PD) runtime system

With a PDL driver, an operation such as reading or writing a complete message starts with an invocation to one of the Tcl entry-points.

Only one operation can be in progress at a time, although it might involve reading and writing several phrases.

This exclusivity is enforced by the driver's state.

Flow of control

PDL driver flow of control is based on the processing of events generated by the device and system itself.

Because of the architecture of the thread system, the programmable driver returns control to the engine when performing a long-running I/O operation.

The Tcl fragments used to construct a PDL driver cannot maintain control across these I/O operations.

States

States are the individual phases of driver function at runtime. During each state, a particular event happens that moves the message process forward or stops it, depending on whether it encounters an error condition.

This table shows the various states:

State	Description
NEW	The thread begins in the NEW state.
IDLE	When initialized, the thread moves from NEW to IDLE and from WRITE to IDLE when the last WRITE function is completed.
READING	The driver reads the inbound message.
WRITING	The driver writes the outbound message.
SHUTTING DOWN	The thread stops.

Write operations

The thread initiates a write operation when it must deliver a message to the system on the other end of a driver's line.

- The device moves to the WRITING state and invokes the Tcl procedure `hci_pd.write`.
- The write procedure uses `hci_pd_send` to set up callbacks when the data is written.
- The `hci_pd.write` procedure sets up for an output (I/O) operation on the driver's device.
- When the Tcl function returns, the PDL writes out the phrase.

When all the data is written or an error happens, the engine invokes the appropriate Tcl continuation procedure specified in the invocation to `hci_pd_send`.

This process may repeat.

When a Tcl fragment returns without initiating an I/O operation, the protocol driver:

- Considers the operation complete.
- Informs the system of the complex operation.
- Returns the driver to the **IDLE** state.

Read operations

Read operations differ from write operations in the way they are started. The engine never explicitly requests to read a complete message. The protocol driver runtime assumes that a read operation should start when data becomes available on the line, the driver is idle, and a phrase is detected.

To the PDL program, the implicit initiation of a read operation looks the same as the explicit initiation of the other operations: the driver moves to the **READING** state and calls the `hci_pd.read` procedure. The protocol driver runtime also initiates a read operation as long as data is available in the input buffer.

To prevent an infinite loop of read operations, which fails to process an input, runtime enforces a requirement that an operation consume any input it receives. For example, using `hci_pd.accept` or `hci_pd.ignore_input`.

Note: If an operation fails to consume the received input, then the driver shuts down in panic mode.

Components of a PDL driver

This list provides basic syntactical rules for coding PDL:

- Every line is closed with a semicolon.
- Double quotation marks mark the beginning and end of a string.
- A forward slash (/) begins and ends a section.
- `/*` starts the comment.
- `*/` ends the comment ("C" style).

Boolean expressions

In some places in PDL, a boolean variable must be supplied.

Select the value from these expressions:

- True condition variable values:
 - `enabled`
 - `yes`
 - `true`
 - `on`
 - `1`
- False condition variable values:
 - `disabled`
 - `no`
 - `false`
 - `off`
 - `0`

Structure of a PDL driver

A PDL driver is composed of two sections:

- [Declarative section](#)
In this section, PDL phrases specify the structure of messages that are transmitted and received through the device. These phrases make up the vocabulary that the driver uses to communicate with its counterpart.
- [Tcl code section](#)
In this section, plain Tcl code defines the entry points for handling protocol-level reads, writes, and other operations.

Declarative section

The Declarative section has these specific definitions:

- Driver definition
- Device definition
- Phrase definition

Driver definition

The structure of a driver definition is:

```
define driver name_of_driver;  
version: "version-string";  
end driver;
```

The `define driver` clause specifies some overall properties of the PDL program not particularly related to its device-driver functionality

`version-string` is user-supplied to identify the iteration of the driver. It is printed with `name`, an identifier, in the system log when the driver is opened by the system.

Device definition

The structure of a device definition is:

```
define device;  
type: device-type;  
attribute...  
attribute...  
...  
end device
```

The `define device` clause declares the device's characteristics that the driver uses. The programmable driver supports TCP drivers structured as clients and servers and asynchronous line drivers.

The `device-type` is required, and is one of:

- `async`
- `tcp-client`
- `tcp-server`
- `decnet-client`
- `decnet-server`

The attributes values depend on the `device-type` choice.

Phrase definition

Phrases are the basic mechanisms that define the communication elements over the device. They define the nouns and verbs that make up a protocol language. Phrases structure read and write operations, and define how to separate the protocol's control characters from the data part of a message.

A phrase is the unit of I/O transfer at the PDL level. Phrase instances are usually composed of many characters, which are units of transfer at the UNIX device level. The characters are composed of several bits, usually 8, which are the units of transfer at the physical level.

The basic structure of a phrase definition is:

```
define phrase name;
phrase-parts
end phrase;
```

A *phrase* represents a way to interpret input, and a framework from which to generate output. A phrase is a hierarchical construction of phrase parts made up of literal text or character classes.

Despite this hierarchical syntactic structure, phrases are interpreted as a linear arrangement of characters and fields. This linear arrangement corresponds to the linear byte-stream that is presented by the UNIX-level device.

A type of composing mechanism is the field. This is the interior node in the hierarchical phrase structure. This acts as a component when writing the phrase, but not when reading the phrase.

For example:

```
# <soh><header><fs><msg data><etx>
define phrase simple-msg;
<soh>
field header = variable-array(not(<fs>));
<fs>;
field data = variable-array(not(<etx>));
<etx>;
end phrase;
```

Extended Backus-Naur Form grammar

The Extended Backus-Naur Form (BNF) grammar for phrase definitions is:

```
phrase-definition ::=
define phrase NAME; phrase-stmt... end phrase;
phrase-stmt ::=
phrase-check {phrase-check-option,... }= phrase-expr ; phrase-check continue = phrase-expr ;
```

```

length-encoded {length-encoded-option,...} = phrase-expr ; field NAME = phrase-expr ; phrase expr;
phrase-expr ::=
variable-array (char-set)
fixed-array (INTEGER, phrase-expr)
begin phrase-stmt... end
char-set
phrase-check option ::=
method: NAME
modulo: INTEGER
store-in: NAME
encoding: encoding
char-set
CHAR
STRING
NAME
not (char-set, ...)
or (char-set, ...)

```

Tcl code section

PDL driver Tcl code is a specific type of programming. Basically, the available Tcl commands for performing I/O operations are commands that set up the operations. They are not represented as commands for performing the operation. The actual operation is deferred to when control returns from the Tcl function. The Tcl invocation chain must be empty at any point where the driver might be required to perform I/O.

To maintain control at the Tcl level, a continuation is passed to the command that sets up the operation. This instructs it to invoke the named Tcl procedure when the operation completes under the appropriate circumstances.

Specific continuations are:

- **ok:** If the operation is successful, then the `ok` continuation is called with a particular argument, depending on the operation.
- **error:** If an error happens, then the `error` continuation is called.
- **timeout:** The `timeout` continuation is called if processes happen too slowly.

Continuation procedures and the driver entry points are always called with a single argument. The argument is a TclX-style keylist containing arguments to the procedure. The arguments are passed as a structured keylist to write forward-compatible driver code.

Note: Do not rely on the exact contents or order of keys of any such keylist argument. Look for the key that is required, and assume its value has the semantics documented there.

Exported Tcl interface

This table shows what functions are required in the Tcl section of a PDL program. It also shows what interface these functions must present to the programmable driver module:

Function		Description
<code>hci_pd.open</code>	<code>info</code>	<p>This function is called when the driver is opened, after the Tcl data structures are initialized, but before anything happens.</p> <p>The <code>info</code> argument is the configuration keylist for the driver.</p> <p>When this (Tcl) function returns, the driver enters the NEW state.</p> <p>If this function fails, then the driver is destroyed and the attempt of the system to open the driver fails.</p>
<code>hci_pd.initialize</code>	<code>info</code>	<p>This is called to initialize the driver, when the driver moves from the NEW state to the IDLE state.</p> <p>Typical functionality for <code>hci_pd.initialize</code> is to open the physical device using <code>hci_pd_open_device</code> and perform any required initialization.</p> <p>Currently, <code>info</code> is empty.</p>
<code>hci_pd.shutdown</code>	<code>info</code>	<p>This function is called to shut down the driver.</p> <p>This procedure is not invoked when the driver is shut down in panic mode.</p> <p>Currently, <code>info</code> is empty.</p>
<code>hci_pd.read</code>	<code>info</code>	<p>This function is called when data becomes available on the line and the driver is in the IDLE state. Under these conditions, it is presumed that a message phrase is arriving over the line, and this function is called to handle message reading.</p> <p>Handling data arriving on the line involves invoking <code>hci_pd_receive</code>.</p> <ul style="list-style-type: none"> <code>hci_pd_receive</code> is the only way to start the PDL parsing the input, although the input can be manipulated in other ways (that is, using <code>hci_pd_ignore_input</code>). Handle the input, even if that means discarding it. Otherwise, the driver terminates on the assumption that it is buggy. <p>Currently, <code>info</code> is empty.</p>

Function	Description
<code>hci_pd.write</code> <code>info</code>	<p>This function is called to write a message on the line from the thread, where <code>info</code> contains the key <code>message</code>.</p> <ul style="list-style-type: none"> The value is the name of the message object to write. This message object is unbound from the Tcl context when the write operation completes. If this operation completes with an error indication, then the engine tries again to send the message. <p>Note: Do not define your own procedures or variables with the <code>hci_pd</code> prefix. This name subspace is reserved for this version and later versions of PDL.</p>

Encodings

The encoding choices for both phrase-checks and length-encodeds are:

```
ascii (radix: radix, width: field-width, \
justification: just, pad: pad-char )
```

- `radix`: is optional. If present, then use 8, 10, or 16, indicating the base choice. If not specified, then base 10 is used.
- `width`: is required. Variable-width-length fields are not currently supported.
- `justification`: is optional. If specified, then use left or right. If not specified, then it defaults to right.
- `pad`: is optional. If specified, then use a character-constant or an identifier which names a single character. Zero (0) is the default pad character.

This encoding represents the value as a number written in ASCII. It can represent numbers in octal, decimal, or hexadecimal notation.

Note: The parameters are specified as comma-separated lists of keywords and values. The arguments are order-insensitive and usually optional.

This table shows encoding examples:

Parameter	Description
<code>ascii (radix: 16, width: 4, pad: 0)</code>	Accepts and generates, for example: 03e8, 00ff.
<code>ascii (radix: 8, width: 10, pad: 0)</code>	Accepts and generates, for example: 1750, 777.
<code>ascii (width: 5)</code>	Accepts and generates, for example: 01000, 19999.
<code>native (bytes: bytes)</code>	This is a binary-encoding, using the computer's built-in byte-order. Set bytes as 1, 2, or 4. Declare the field as fixed-array (<code>bytes, any</code>), where the field's bytes is the same as bytes in the built-in "(" declaration.

Parameter	Description
network (bytes: bytes)	This binary coding is basically the same as the built-in, except it uses acknowledged (<i>network</i>) byte order instead of the built-in byte order.

Phrase-constant parts

One type of phrase part is a constant. Phrase-constants are written with literal characters, strings, named characters, or character classes. Character classes are phrase-constant parts that can match one of several different characters on input.

This table shows examples of phrase-constant parts:

Phrase-constant part	Description
X	Matches the character X in the input, and generates an X character on output.
XYZ	Essentially equivalent to 'X'; 'Y'; 'Z'.
<ack>	Matches the ACK character (decimal value 6) on input, and generates one on output.
letter	Matches any letter (A-Z, a-z) on input. It cannot display in a writable phrase except within the body of a phrase part.

Note: In the current implementation, you protect a character class by a field phrase part.

Fixed-array parts

A fixed-array denotes a portion of the phrase that is made up of a fixed number of repetitions of a particular form.

```
fixed-array (count, component)
```

- `count` is an integer constant.
- `component` is any phrase part, even a complex phrase part.

A field cannot be inside of a fixed-array. Large fixed-arrays may be inefficient unless `component` is `any`. For example, if the protocol calls for a block of 1000 digits, consider using `fixed-array (1,000, any)` even though `fixed-array (1,000, digit)` is more exact.

This table shows examples of fixed-array parts:

Fixed-array part	Description
<code>fixed-array(1024, any)</code>	A 1K block of arbitrary (that is, potentially binary) data.
<code>fixed-array(5, digit)</code>	A 5-digit number.
<code>fixed-array(10, begin ot(<nl>)); <nl>; end)</code>	<code>variable-array(n</code> 10 newline-terminated lines.

Variable-array parts

A variable-array phrase part describes a message part that is made up of a variable number of characters. Variable-arrays are restricted to the cases. By looking at each succeeding character in isolation, it can be determined if it belongs in the variable array. This property is reflected by the fact that `component` is a character set.

On input, the variable array matches as many successive characters in the `component` character set as possible.

- `component` can be more than a single character. It can be a character constant, a named character set, or a composition of a character set using `not` or `or`.
- If `component` is `any`, then this form has special meaning. Use this form inside a length-encoded region.

Except for length-encoded phrase parts, use variable-arrays to read a variable-length field up to some terminating character.

For example:

```
define phrase a-phrase;
field line = variable-array(not(<cr>));
<cr>;
<lf>;
end phrase;
```

Note: Do not use a variable-length field at the end of a phrase. The PDL would not know when to stop trying to read characters for the field.

Fields

Fields name parts of a phrase for content manipulation in various ways.

```
field name = contents
```

- `name` is an identifier that refers to the field.
- `contents` is any phrase structure, except that it cannot contain another field.

A field is used to:

- Delineate part of the data portion of the message.

- Indicate where outbound data is interpolated.
- Store a length-encoding.

Store an encoded phrase-check value. When a phrase is read (set up using the `hci_pd_receive` Tcl command), the corresponding continuation is called with a keylist. The keylist associates the field names with the input region that contains their value. When a phrase is written, the engine uses the field's value without regard to its content structure.

The restrictions on the field contents during reading do not apply when writing; you must ensure the protocol is satisfied.

Phrase-checked parts

PDL has built-in support for protocols that employ checksums or block-check characters.

```
phrase-check {method: method,
modulo: mod,
encoding: encoding,
store-in: field name } =
begin
body
end;
```

Phrase-checked part	Description
phrase-check part	Only one is permitted in a phrase definition, although it can be continued.
phrase-check	This continued phrase part is used to run a phrase check over disjoint phrase portions.
method:	<p>This is required to specify how the phrase-check value is computed as a function of the characters in the phrase-check region.</p> <p>These identifiers are available to specify the <code>phrase-check</code> method:</p> <ul style="list-style-type: none"> • <code>xor</code> exclusive-ors the character codes of the checked characters to form the phrase-check value. The phrase-check value ranges from 0 to 255. • <code>add</code> adds together the character codes of the checked characters.
modulo:	<p>This applies an arithmetic modulo operation to the phrase-check value before it is passed to the encoder. If supplied, then use an integer constant for <code>mod</code>.</p>

Phrase-checked part	Description
encoding:	This specifies how the phrase-check value is converted into a string that is placed in the data stream, or extracted from the data stream. It is a required keyword, and must specify a valid encoding choice.
field name	This must not name a field that is inside body.

There is no conceptual upper bound on the phrase-check value. This implementation can only represent numbers up to about 500,000,000, and overflows at 2Mb of 0xFF's.

This example shows a disjointed phrase-check region:

```
# <data>, <data>, CHECK=<5-byte checksum>
define phrase split-checksummed-msg;
phrase-check {store-in: chk,
method: add,
modulo: 65536,
encoding: ascii( width: 5 ) } =
begin
field line1 = variable-array(not(',',));
end;
',';
phrase-check continue =
begin
field line2 = variable-array(not(',',));
end;
',';
"CHECK=";
field chk = fixed-array(5,digit);
',';
end phrase;
```

If this phrase is written with line1 that is bound to `abc` and line2 that is bound to `123`, then this transmits:

```
abc, 123, CHECK=00444
```

This is because the sum of the ASCII codes for a, b, c, 1, 2, and 3 is 444 decimal.

Length-encoded phrase parts

The programmable driver supports length-encoding of line phrases.

The syntax is similar to that for phrase checks:

```
length-encoded {encoding: encoding,
store-in: field-name } =
begin
body
end;
```

The READ semantics are that the phrase part matches if the body matches, where:

- The choices for `encoding` are the same as those for the `phrase-check` phrase part.
- `field-name` indicates the body length, in bytes.

If the `body` contains a variable-array (any) subpart, then the variable-array's actual length is the length stored in `field-name` less the length of all the body's fixed-length parts.

When writing a command, reverse computation happens. The `field-name` is completed with the actual length of the field text with a structure of variable-array (any) structure plus the length of all the body's fixed-length parts.

- For these operations to make sense, the `body` can contain only zero or one variable-array (any). All other fields inside `body` must be a fixed length.

In particular, if `field-name` happens inside of `body`, then it must use a fixed encoding. The compiler verifies that these conditions hold.

For example, in a typical 2-byte length-encoded phrase that is used on a TCP connection:

```
# <2-byte length data>
define phrase blob-data;
length-encoded {encoding: network( bytes: 2 ),
store-in: len } =
begin
field len = fixed-array(2,any);
field blob = variable-array(any);
end;
```

Available Tcl functions

This section specifies which commands are available in the Tcl section of a driver program. Those functions which are particular to PDL programs are also available. These are the usual Tcl and hci extension commands.

hci_pd_receive contns

This function prepares to parse the input off the line. It attempts to match the input against each of the given phrases. The `contns` argument is a keylist that instructs the programmable driver what to do in various circumstances. Apart from special keys, the mapping is from names of phrases to Tcl functions that are called when the named phrase is read by the system.

This table shows keys in `contns` which are treated specially and not as phrase names:

Key	Description
no-match	<p>Specifies the continuation to invoke when the received input does not match any of the specified phrases.</p> <p>If a match is found, then the corresponding Tcl function (continuation) is called.</p> <p>no-match continuation is called.</p> <p>If, in trying to match the input, an I/O operation times out, then the time-out continuation is invoked.</p>

Key	Description
<code>line-error</code>	<p>Invokes the continuation if the connection is lost. This is loss of carrier signal for async lines with <code>require-cd</code> enabled, or remote closure of a socket for TCP servers or clients. The continuation is also called if parity or a framing error is detected, or there is an unexpected end of input.</p> <p>Use this key for TCP devices if the remote process dies, or, in the case of an async modem line, if the carrier is lost.</p>
<code>error</code>	<p>Specifies that both <code>no-match</code> and <code>line-error</code> are the same continuation. Do not use it with those keys.</p>
<code>timeout</code>	<p>Specifies the time-out length and is a required procedure to invoke (a Tcl continuation to resume) in the event of time-out.</p> <p>Make the value associated with the time-out key a Tcl list of two elements.</p> <p>The first is an integer which specifies the length of the time-out, in milliseconds.</p> <p>The second is the usual Tcl continuation.</p> <p>If omitted, then the operation can take arbitrary time.</p>

If an error continuation for `no-match` or `line-error` is not specified and `error` is not given, then the engine uses one of these default functions:

- `no-match`
Discards all of the buffered input and returns to the IDLE state.
- `line-error`
Signals an error and terminates the driver.

All of the continuations are passed a keylist that contains a `status` key.

For a successful read, the value of `status` is `ok`. The keylist that is passed to a successful read continuation reads the field names of the phrase as keys. Each key has a value that is the field value region of the input buffer. This keylist also contains the key `end`; its value is the region of the input buffer which was not read to match the phrase.

For an error continuation, the `status` indicates the type of error:

- `no-match`
- `line-error`
- `timeout`

Example

```
hci_pd_receive {{foo did_read.foo} {no-match failed} \
{timeout {1000 tooslow}}}
```

Use this example to match a `foo` phrase with these options:

- Calling the Tcl procedure `did_read.foo`, when successful.
- Calling the Tcl procedure `failed`, when the received data did not match the `foo` phrase.
- Tolerating up to 1 second between characters.

hci_pd_send phrase data contns

This writes a phrase onto the line, where:

- *phrase* is the name of the phrase to write.
- *data* is a keylist that maps the phrase's field names to values to write.
- *contns* is a keylist describing what to do when the write completes.
contns should minimally provide continuation procedures for `ok` and `error`. These continuations are called, respectively, on successful write and some kind of device error processes. The continuations are passed an empty argument keylist.

Timeouts are not supported for write operations. PDC 3 passes a status key to the continuations, and a type key to the error continuation, with a value of device or format.

Example: foo phrase

```
define phrase foo;
"-->";
field n = variable-array(digit);
",";
field d = variable-array(not(<cr>);
<cr>
end phrase;
```

In using the `foo` phrase, you can cause an instance of this phrase to write to the device. The `d` field is completed with the contents of the message object named `message3`. The `n` field completed with the literal string `001` using:

```
hci_pd_send foo {{d {message message3}}
{n {literal 001}}} \
{{ok wrote_fine} {error write_error}}
```

Typically, the message comes from the value associated with the `msg` key in the argument to the `hci_pd.write` procedure.

If the data contents of `message3` are `Attack at Dawn`, then this is written to the device:

```
-->;001;Attack at Dawn \ r
```

\ r denotes the ASCII character with code 13 decimal (xod hex).

The typical usage extracts the message name from the arguments to the function.

Example: Using sequential numbers

```
proc hci_pd.write {msg} {
  global seq_num
  # increment the current sequence number
  set seq_num [expr $seq_num + 1]
  # send a packet with the new sequence number
  hci_pd_send in_sequence_phrase \
    [list [list header [list literal \
    [format "%03d" $seq_num]]] \
    [list data [list message \
    [keylget msg message]]] \
    [{ok wrote_ok}] }
```

The Tcl environment provides the `format` function.

hci_pd_accept info

This arranges to deliver the specified portions of the input as a message to the system, where *info* is a Tcl keylist with these keys:

- `text` is required, and is associated with a list of regions to extract from the input as the data portion of the message.
- `end` is required, and its value is the amount of input to extract. The end key value is a list of two numbers, with the first being the amount of input to extract.
- `code` is optional, and is the read completion code to pass to the system along with the data part of the message. 0 usually indicates a successful protocol-level read. If omitted, then the value defaults to 0.

Example

The input buffer contains:

```
*100*Me...*
```

- `*` are protocol control characters for this protocol.
- `100` and `Me...` are data fields matched with this phrase:

```
define phrase record;
  "*";
  field id_num = fixed-array(3,digit);
  "*";
  field name = fixed-array(5,any);
  "*";
end phrase;
```

You can then specify this command to accept the input as a valid message and deliver it to the engine:

```
hci_pd_accept {{text {{1 3} {5 5}}} end 11}
```

Usually, it is unnecessary and undesirable to code the text regions directly.

If the results of the phrase-match are stored in the variable `info`, then specify:

```
hci_pd_accept "{{text {[keylget info id_num]}
               {[keylget info name]}}}
               {end [keylget info end]}}"
```

This shows the input buffer's state after receiving two complete messages in the `foo` phrase form that is given above:

```
-->;001;Attack at Dawn\r-->;002;Retreat\r
|||||
```

If `hci_pd_receive` was called with `foo` as one of the phrase arguments, then the `phrase-match` and its continuation are called with this keylist:

```
{{status ok} {n {3 3}} {d {7 14}} {end {22 15}}}
```

The regions have a zero origin. The `end` key describes where the `foo` phrase ends, at offset 22. It also describes how much data is in the input buffer after that (15 characters).

This shows the basic reason for the `end` key to `hci_pd_accept`. Remove only the portion of the input buffer that corresponds to the message being accepted. The remaining portion of the input buffer can contain other messages.

In this case, `hci_pd_accept` is called with:

```
{{text {{7 14}}} {end 22}}
```

The first message is extracted from the input buffer and sent to the engine. In this case, the only required operation is to accept the one phrase. The PD module automatically calls again when it sees there is still data in the input buffer after the current READ operation completes.

hci_pd_deliver

The usage for this command is `hci_pd_deliver data code` (code is required and usually 0). Data must not contain nulls.

This is used in lieu of the `hci_pd_accept` command to return data and a code, usually 0, to the system engine. If this command is used, then the data in the inbound buffer must be exhausted by the `hci_pd_ignore` command.

```
hci_pd_ignore_input
hci_pd_ignore_input len
hci_pd_ignore_input -all
hci_pd_ignore_input -until target
```

This discards input from the beginning of the input buffer. Specify the amount of data to discard in one of these ways:

- A fixed-length prefix is discarded.
- All of the input is discarded.
- All of the input up to, but not including the character target, is discarded.

The possible formats for the target argument are a single, non-backslash character by itself. It can also be a backslash followed by a number (0-prefix for octal, x-prefix for hex).

These targets are equivalent:

- A
- \0101
- \x41
- \65

Currently, these are the only character formats understood by the third form of `hci_pd_ignore_input`.

Discarding input changes the data underlying regions of the input buffer. Because field extents are represented as input regions, using this command changes the data underlying a particular region.

For example, suppose you have recognized a message and therefore know that field `foo` happens in region `{5 5}` of the input. If the first three characters of the input are discarded using any of this command's forms, then the field now happens in region `{2 5}` of the input.

If an attempt is made to access the contents of field `foo` without taking into account the discarded prefix, then the incorrect data is given.

This function returns the amount of data that remains in the input buffer. In the case of the `-all` option, zero (0) returns.

If the amount to discard is more than exists in the input buffer, then the system discards the entire input buffer and does not signal an error (zero returns).

This might happen, for example, if the target specified with the `-until` option does not happen in the input buffer. It could also happen if `len` is greater than the amount of data in the buffer.

Examples

These examples show the common uses of `hci_pd_ignore_input`.

To discard all the buffered input when a failure to match any phrase happens:

```
proc hci_pd.read {info} {
  hci_pd_receive {... {no-match forget-it}}
}

proc forget-it {info} {
  hci_pd_ignore_input -all
}
```

To scan for a recognizable start-of-message character when a phrase-match failure happens:

```
define phrase foo;
  "[";
  field data = variable-array\
    (or(letter,digit,<cr>));
  "]"
end phrase;
```

```

proc hci_pd.read {info} {
  hci_pd.receive {... {no-match bummer}}
}
proc bummer {info} {
  hci_pd.ignore_input -until {} }

```

hci_pd_open_device [contns[config]]

This establishes the connection between the driver and the operating system. It also prepares the device for use, where the *config* argument shadows the declarations made by the driver's `define device` statement.

The keys are the same as the keywords in the `define device` construct, except that lower-case letters are upper-case.

Example 1

The device definition is:

```

define device;
type: async;
baud: 1200;
end device;

```

With this definition, you can open the device with a different line speed by calling:

```

hci_pd_open_device {} {{BAUD 9600}}

```

The keys that are given in configuration override the specifications made by `define device` on a per-key basis.

In the case of a TCP server or an async device with `require-cd` enabled, this function waits for the client to connect before calling the `ok` continuation.

As for `hci_pd.receive`, specify a time-out. If the connection fails to establish within the specified time, then the time-out continuation is called.

These continuations, similar to others in this interface, are called with a single keylist argument. The keylist contains a status key with a value of the type of continuation invoked (for example, `ok`, `timeout`, `temp-error`, or `perm-error`).

These are error continuations apart from the time-out continuation:

- `temp-error` is called when the system detects a transient error, such as when the host is unreachable for a TCP client.
- `perm-error` is called when an error condition arises that does not go away. For example, an error in the configuration string, or an unknown host name.

An error key is shorthand for specifying both `temp-error` and `perm-error`.

Example 2

This example shows the use of `hci_pd_open_device` with continuations and overrides.

In this example, the driver's exact configuration is not fully known at compile time. The path to an async device file is taken from some driver-specific configuration information stored in a Tcl global variable.

```
proc hci_pd_open {info} {
  global some_config_info
  hci_pd_open_device {{ok open_ok} \
  {error open_failed}} [list [list PATH \
  [keylget some_config_info my_path]]]
}
proc open_ok {info} {
  hci_pd_send welcome_phrase...
}
proc open_failed {info} {
  hci_pd_set_result_code 2
}
```

hci_pd_report_exception code text

This reports an exceptional condition to the engine, where *code* is any integer and *text* is arbitrary text.

For example, to log certain received messages for the system operator:

```
proc hci_pd.read {info} {
  hci_pd_receive {{normal-msg handle_normal}\
  {powering-down pwr_down}}
}
# receive a "power going down" message from the\ other end
proc pwr_down {info} {
  hci_pd_report_exception 3 \
  {Remote device is powering down!}
  hci_pd_ignore_input [lindex [keylget info end] 0]
}
```

The use of `hci_pd_ignore_input` to remove the text of the powering-down phrase from the PD's input buffer. This is necessary to avoid having the PD stop the driver on the assumption it was malfunctioning because it did not process its input.

hci_pd_set_result_code code

This communicates an operation's completion status to the system. *code* is an integer that is passed to the engine when the current operation completes, when the state transitions back to IDLE or DEAD.

In general, non-zero values denote error conditions, and 0 denotes a successful operation.

Use this command to clear (set to zero) the result code when a low-level error happens. The PDL runtime sets the result code to a non-zero value when a low-level error happens. When this happens, the default sends an error to the engine.

Example

A typical use for this function is to set a non-zero completion code when an operation fails.

In this example, the system interprets a result code of three (3) when an output error happens.

```
...
hci_pd_send some_phrase {...} {... {error oops}}
...
proc oops {info} {
  hci_pd_set_result_code 3
  # returning here completes the operation, in
  # this case with a completion code of 3
}
```

uassert expr [descr]

This function is useful for verifying that a condition holds.

If *expr* denotes a false value in Tcl style, then the assertion fails. In this case, the driver shuts down in panic mode.

If *descr* is supplied, then it is part of the diagnostic message logged by the system when the driver shuts down.

For example:

```
...
uassert [expr $int_seq == $extn_seq] {Out of sync}
...
```

Built-in PDL functions

Many protocols are similar in operation, and differ only in the delimiter characters that are used. To simplify handling of these common cases, built-in PDL functions are included that implement most of the definitions and Tcl code for the driver.

To use a function, define only those elements unique to your protocol needs. For example, data phrase definitions.

To specify a style to use in your driver, use the `hci_pd_msg_style` command.

For example:

```
hci_pd_msg_style style key:value ?...?
```

The system supply two message styles: `basic` and `acknak`.

The `basic` style sends and receives phrases.

This table shows the configuration keys and values:

Key	Description
phrase: <i>phrase-name</i>	Specifies the name of the phrase to use to encode and decode the message. This entry is required.

Key	Description
field: <i>field-name</i>	Specifies the name of the field from the phrase into/from which the message data is placed/obtained. This entry is required.
dir: <i>direction</i>	Indicates the direction in which the driver operates. Choices are <code>bidir</code> (bidirectional) and <code>obonly</code> (out-bound only). This entry is optional and defaults to <code>bidir</code> .
resync: <i>character</i>	<p>Makes the driver use resynchronization logic if an error or time-out happens. To resynchronize, the driver scans the input buffer the next instance a character is read.</p> <p>Not specifying this entry discards all data in the input buffer if an error or time-out happens. This entry is optional and defaults to 15040.</p>
timeout: <i>microseconds</i>	Specifies the general time-out. This is used for both read and write operations; it is expressed in microseconds. This entry is optional and defaults to 15040.
rtimeout: <i>microseconds</i>	Specifies the read-specific time-out value. This entry is optional and defaults to 15040.
wtimeout: <i>microseconds</i>	Specifies the write-specific time-out value. This entry is optional and defaults to 15040.
acknak	<p>This style uses all of the keys in the <code>basic</code> style, plus these:</p> <ul style="list-style-type: none"> ackphrase: <i>phrase-name</i> Specifies the name of the phrase that recognizes ACK messages. This entry is required. nakphrase: <i>phrase-name</i> Specifies the name of the phrase that recognizes NAK messages. This entry is required. naktries: <i>count</i> Specifies the number of NAK responses before giving up on writing a message. This entry is optional and defaults to zero (0). tmotries: <i>count</i> Specifies the number of time-out events before giving up on writing a message. This entry is optional and defaults to zero (0).

Named character classes

To make writing phrases more convenient, the PDL program provides PDL-specific named character classes for certain characters and character classes. Classes are a set of characters which are equally acceptable.

This table shows the available character classes:

<nul>: 00	<soh>: 01	<stx>: 02	<etx>: 03	<eot>: 04	<enq>: 05	<ack>: 06
<bel>: 07	<bs>: 08	<ht>: 09	<dle>: 10	<dc1>: 11	<dc2>: 12	<dc3>: 13
<dc4>: 14	<nak>: 15	<syn>: 16	<etb>: 17	<can>: 18	: 19	<sp>: 20
<nl>: 0a	<lf>: 0a	<vt>: 0b	<np>: 0c	<cr>: 0d	<so>: 0e	<si>: 0f
<sub>: 1a	<exc>: 1b	<fs>: 1c	<gs>: 1d	<rs>: 1e	<us>: 1f	

The pre-defined character classes are:

- `digit` is an ASCII digit, 0...9.
- `uppercase letter` is an uppercase ASCII letter, A...Z.
- `lowercase letter` is a lowercase ASCII letter, a...z.
- `letter` is an ASCII letter, A...Z or a...z.
- `alphanumeric` is a letter or a digit.
- `any` is any character.

Note: Single character names are enclosed in angle brackets (<...>). Character classes are plain identifiers.

Sample PDL device definition for a TCP driver

TCP clients and servers basically look the same. The only difference in addition to the `type: value` is that a TCP client supplies a value for `host`:

For example:

```
define device;
type: tcp-client;
port: port-num;
host: server-host
end device;
```

- For TCP drivers, `port-num` specifies the TCP port number for the connection. This must be an integer constant or an identifier which names a TCP port. For example, `/etc/services`.
- For TCP clients, `server-host` indicates to which machine a connection is attempted. In this case, use an identifier or string.

Example

The device definitions for a driver pair might resemble this:

- The server, on host `foo.bar.com`:

```
define device;
type: tcp-server;
port: 4401;
end device;
```

- The client, elsewhere:

```
define device;
type: tcp-client;
port: 4401;
server: "foo.bar.com";
end device;
```

- If the client is in the same domain and port 4401 is known by the name `the service`, then:

```
define device;
type: tcp-client;
port: the service;
server: foo;
end device;
```

Note: In PDL keyword-value type constructs, supply an identifier where a string is expected. In this example, this is the value of the host and port name.

Sample PDL device definition for an async driver

This sample driver shows the structure and components of an async driver:

Structure	<pre>define device; type: async; path: device-path; baud: baud-rate parity: parity-mode data-bits: data-bits; stop-bits: stop-bits; xon-xoff: sw-flow-flag hw-flow-control: hw-flow-flag require-cd: cd-mode flag end device;</pre>
-----------	---

Options for async drivers

Option	Description
<code>path: device-path</code>	Identifies the path to the UNIX device file that represents the async line to drive. This is a required attribute.

Option	Description
baud: <i>baud-rate</i>	<p>Uses an integer constant from this list:</p> <ul style="list-style-type: none"> • 50 • 75 • 110 • 134 • 150 • 200 • 300 • 600 • 1200 • 2400 • 4800 • 9600 • 19200 • 38400 <p>If omitted, then <code>baud-rate</code> defaults to 9600.</p>
parity: <i>parity-mode</i>	<p>Specifies the available parity-modes: even, odd, and none. If the parity is not specified, then it defaults to none.</p>
data-bits: <i>data-bits</i>	<p>Specifies the number of data bits.</p> <ul style="list-style-type: none"> • If <code>data-bits</code> is specified, then it must be 5, 6, 7, or 8. • If a number is not specified, then it defaults to 8.
stop-bits: <i>stop-bits</i>	<p>Controls the number of stop bits in the async frame and is an integer constant (1 or 2). If an integer constant is not specified, then it defaults to 1.</p>
xon-off: <i>sw-flow-flag</i>	<p>Determines whether to use software flow control (that is, <code>xon/xoff</code>).</p> <ul style="list-style-type: none"> • If <i>sw-flow-flag</i> indicates a true value, then software flow control is enabled. • If not specified, then it defaults to disabled.
hw-flow-control: <i>hw-flow-flag</i>	<p>Controls whether to use hardware flow control (RTS and CTS signals).</p> <ul style="list-style-type: none"> • If <i>hw-flow-flag</i> indicates a true value, then hardware flow control is used. • If not specified, then it defaults to disabled.

Option	Description
<code>require-cd: <i>cd-mode-flag</i></code>	<p>Controls whether CD (Carrier Detect) is used and required.</p> <ul style="list-style-type: none"> If <code>cd-mode-flag</code> is a true value, then carrier detect is required. In this case, a loss of carrier at any time generates a line error. The invocation to <code>hci_pd_open_device</code> does not complete until it detects a carrier. This is appropriate when connected to a modem, but is also useful in other hardware circumstances. If <code>cd-mode-flag</code> is a false value, then the carrier detect signal on the line is ignored. The open completes without it, and losing the signal does not indicate a lost connection. If a flag is not specified, then it defaults to <code>no</code>.

Example

For example, a typical device declaration for driving `tty0` would look similar to this sample:

```
define device;
type: async;
path: "/dev/tty0";
baud: 9600;
end device;
```

Many options can default to the commonplace values.

Compiling PDL code

To compile a PDL, use the `hcipdc` command.

Store PDLs in the `pd1` directory, with a `.pd1` extension.

```
PDL basic style example
mlp_tcp.pd1
define driver tcpip-mlp;
version: "2.0";
end driver;
/* This driver manages the transmission of messages
 * using the HL7 defined MLP protocol. Each message
 * is bounded by a start character <0xb> and a stop
 * string <0x1c><0xd>.
 *
 * The phrase basic-msg recognizes this message
 * format.
 * Once recognized, the message data will be
 * available from the 'data' field.
 */
define phrase basic-msg;
<vt>; field data = variable-array( not( <fs> ) );
<fs>;
```

```

<cr>;
end phrase;

/*****
End of declarative section, TCL management functions start here.
*****/
##
# This can be handled completely using the "basic" style.
hci_pd_msg_style basic phrase:basic-msg field:data\
                    resync:\0xb
Async PDL Example

##
msmeds_async.pdl
define driver msmeds;
version: "2.0";
end driver;
/* This driver manages the transmission of messages * to the MSMeds system.
* Protocol Description:
*   Data Format: ASCII
*   Character Format:
*   Length: 1 start bit
*           7 data bits
*           1 stop bit
* Parity: odd
*
* Record Format:
*   Start Text STX (ASCII 002, Hex 0x02)
*   Text
*   End of Text ETB (ASCII 027, Hex 0x17)
*   BCC 3 ASCII characters (see below for
*   calculation)
*   End of Message ETX (ASCII 003, Hex 0x03)
*
* ACK/NAK: ACK (ASCII 006, Hex 0x06 )
* NAK (ASCII 025, Hex 0x15 )
*
* Timeout: Wait 30 seconds for ACK, then timeout
*
* The BCC calculation is a modulo 128, performed by
* adding all characters from the Start of Text to
* the End of Text inclusive. The resulting unsigned * value is then divided by 128 and the re
maining
* 8-bit quantity is converted to 3 ASCII digit
* characters and added to the source message.
*
* <stx><message text><etb><bcc1><bcc2><bcc3><etx>
*
* o If the message is ACK'ed, we are done
* o If the message is NAK'ed, mark message as NAK'ed
*   and return failure
* o If a timeout happens, mark message as timeout and
*   return failure
*/
define device;
type:async;
path:"/dev/tty1";
line-speed:9600;
stop-bits:1;
start-bits:1;
data-bits:7;
parity:odd;
xon-xoff:disabled;
hw-flow-control:enabled;
require-cd:enabled;
end device;
/*Define the basic message structure. Set up the bcc
* calculation to include the <stx>, data, and <etb>
* characters.
*
* BCC calculations are handled using a PDL concept
* called a virtual field (in this case, phrase-
* check). Virtual fields are automatically
* constructed if you do not specify data for them in

```

```

* the write operation. In this case it
* auto-calculates the BCC value and \ places it
* into the BCC field.
*/
define phrase basic-msg;
<stx>;
phrase-check { method: xor, store-in: bcc } =
begin   field data = variable-array( not(<etb>));
end;
<etb>;
field bcc = fixed-array( 3, digit );
<etx>;
end phrase;
/* You must define each message you write/read as a
/* phrase. Define ACK and NAK.
*/
define phrase ack-msg;
<ack>;
end phrase;
define phrase nak-msg;
<nak>;
end phrase;
/* Define a phrase for ignoring inbound data */
define phrase any-byte;
field junk = fixed-array( 1, any );
end phrase;
/*****
End of declarative section, TCL management functions start here.
*****/
#{#
hci_pd_msg_style acknak phrase:basic-msg \
field:data      \
ackphrase:ack-msg \
nakphrase:nak-msg \
timeout:10000    \
naktries:3       \
tmotries:3
#}#

```


Cloverleaf multi-byte encoding

Multi-byte encoding permits the system to effectively use unicode character encoding. With multi-byte, the data streams are treated as Unicode code points. Because Unicode is a superset of all other character sets, the system can handle any external character set. It provides for conversion between the external character representation and Unicode in the protocol threads.

Because both Java and Tcl have Unicode support, the approach is uniform throughout the system.

In the engine, the unicode representation is the UTF-8 encoding due to its intrinsic compatibility of representing ASCII as unmodified single bytes. This provides a high level of backward compatibility with existing versions. The message data arriving at or sent from the engine can be any codepage, including UTF-16. (The system engine provides user-selected translation of the encoding to UTF-8 for internal engine processing.)

Multi-byte is not intended to provide a full multilingual user interface because the command-line utilities are based on the locale codepage. The codepage usually does not support mixing languages beyond ASCII and the locale. Multi-byte transports and transforms data messages in different languages such as English, French, Arabic, and Chinese in their respective OS environments. The GUI for development, administration, and monitoring of the system can be in English. All data strings in the GUI are displayed properly in the different languages.

For a particular language, different messages can be encoded differently and support a variety of codepages for both incoming and outgoing data messages.

You can specify an external encoding for each thread. The specified encoding applies to both inbound and outbound messages for that thread.

You can choose to use a byte order mark. If a byte order mark is found in the inbound data stream, then the encoding is determined automatically.

You can choose the XML encoding mode. The XML mode determines the encoding dynamically from an XML declaration. XML encoding works for inbound and outbound messages. XML encoding and **Use byte order mark** can both be specified for inbound messages. In this case, a byte order mark takes precedence if found. If no byte order mark or XML declaration is found, then the XML mode assumes UTF-8 encoding.

You can choose to generate a byte order mark. This applies to outbound messages. Byte order marks are generated for Unicode UTF encodings, that is, UTF-8, UTF-16, and so on. If no encoding options are specified, then binary encoding is assumed. Binary encoding is an identity mapping; that is, byte values 00-FF hex translate into unicode code points 00-FF. Pre-existing sites that are upgraded use the default binary encoding to maintain existing behavior.

Backward compatibility includes existing configuration files, engine behavior, and system resource consumption.

The operation of existing sites is unchanged. Any incompatibilities with existing files are documented with conversion procedures for migration of existing sites.

Some performance degradation is inevitable because there is more data to process for each character. Sometimes increased internal efficiency provides a performance increase.

Unicode is supported in the user interface, permitting display of foreign characters and entry of foreign characters on the keyboard. Foreign characters are permitted in object names where practical. This includes file names, directory names, and so on. Foreign characters are permitted in file names used for data transfer

Process names must be ASCII.

Encoding conversions are bypassed for inbound and outbound messages as a performance enhancement. The engine stores data internally as UTF-8. Therefore, threads that use UTF-8 or ASCII data can use bypass encoding to eliminate the performance penalty of encoding conversion. A negative effect of using bypass encoding is that the input data is not validated, so invalid byte streams can get into the engine. This does not cause any serious problems, but invalid messages may error out inside the engine.

The use of bypass encoding is basically your guarantee that the data is UTF-8 or ASCII and encoding conversions can be skipped for increased performance. The use of UTF-8 or ASCII as the encoding performs a conversion. Even though a UTF-8 or ASCII encoding conversion does not change any data, it invokes the associated validation processes.

Inbound encoding conversion errors can make it impossible to convert the inbound data to Unicode. It is therefore necessary for inbound SMAT and recovery state 1 to be raw data. Then messages with encoding errors can be examined, corrected, and resent in a consistent fashion.

An on-screen keyboard is available with most operating systems to assist with entering file names and text that use different characters.

Limitations of unicode

These are the limitations of unicode:

- PDL start and stop bytes cannot happen in multi-byte characters.
Multi-byte refers to the encodings that are used outside of the system. Control characters meet this requirement for most encodings, so they are usually not a problem.
- Thread names, site names, and process names must be ASCII alpha-numeric. This is due to limitations in the Raima version that is used.
- Separator characters for VRL, HL7, X12, and similar message formats must be ASCII. Segment names must also be ASCII.
- Unicode characters are limited to the 16-bit Basic Multilingual Plane (BMP).
Unicode version 3 introduced a 31-bit code space comprising many supplemental planes. Tcl only supports the 16-bit code space and Java only adds support for supplemental characters in version 5. The 31-bit code space cannot be used until the Tcl and Java issues are resolved.
- Object names that interact with advanced security server may be limited to ASCII.
- The FRL subfield prefix must be ASCII.

- 1.2.10. Extended ASCII backward compatibility limitation.
- European characters that have a value greater than 127 become 2-byte UTF-8 characters. For those existing implementations that contain extended ASCII, this might not be backward compatible.

Remote machines in a different operating system

The default encoding is UTF-8 on a Linux terminal/console.

The Windows telnet does not support UTF-8. When the Linux server is configured as UTF-8 (default), you must have a UTF-8 compatible telnet client.

Handling of unsupported encodings

If an unsupported encoding is specified in the NetConfig file, then the thread enters an error state immediately during startup and no messages are processed. The GUI does not permit entry of an invalid value, so this should be extremely rare.

If XML encoding is used and an unsupported encoding is encountered in an XML declaration, then the offending message is transferred to the error database. This encoding is given error state 416: Inbound encoding conversion error. In this case, the data that is stored in the error database is the raw byte stream that was received. Error state 416 is the only time that raw data is stored in the error database. All other errors are stored in UTF-8 Unicode.

Handling of invalid byte streams

Many encodings place restrictions on which byte sequences are acceptable. For example, Big5 requires that the first byte is from A1h to FEh. The second byte must be from 40h to 7Eh or A1h to FEh. If an inbound byte stream is not valid for the encoding in use, then the message is transitioned to the error database with error state 416: Inbound encoding conversion error.

These messages are unintelligible and cannot be converted to Unicode, so the raw byte stream is stored in the error database. Outbound messages always consist of valid byte streams. Byte stream validity for an encoding is a separate issue from the ability of an encoding to represent all the Unicode characters in a message.

msgfinduchar: Maximum permissible character value

VRL, HL7 and other delimited formats are restricted to having ASCII delimiters. You can use `msgfinduchar` to find an unused ASCII character in a message to act as a new delimiter.

ASCII is defined as a 7-bit character and this range maps directly to Unicode code points that are a single byte in UTF-8. Non-Unicode characters above this range depend on the codepage used for pre-Unicode input. The translation to UTF-8 results in an unknown ordering of code points. This reordering makes the "greater than" search of the command a problematic operation.

To flag and avoid this problem, the command produces an error if the input character is greater than or equal to 127.

Non-ASCII data converted to UTF-8

Existing configuration files containing non-ASCII data (byte values above 7F hex) are converted to UTF-8.

A site upgrade utility automatically converts standard configuration files. A binary to UTF-8 conversion utility permits you to convert any non-standard configuration files.

These conversion utilities first scan input files:

- If a file contains only ASCII, then no conversion is required.
- If a file contains byte values from 80-FF hex and is not valid UTF-8, then it is converted.
- If a file contains byte values from 80-FF hex and it is valid UTF-8, then it has probably already been converted. In this case, a message is given: `file xxx is already UTF-8, do you wish to convert it anyway?` You have the option to convert or skip. This reduces the risk of running the conversion twice and double-converting files.

These configuration files are automatically converted:

- `NetConfig`
- `tclprocs/*`
- `Tables/*`
- `Xlate/*`

A behavior change happens for XML processing. In previous versions, a dynamic encoding conversion was performed during XML-to-XML translation if the encoding was changed by modifying `?xml.&encoding`. This conversion no longer happens because all is internally normalized to unicode. Upgraded protocol threads are binary encoding so there is no conversion in the threads. This can result in different behavior for sites that use an XML-to-XML translation where the encoding is altered in translation.

To compensate for this behavior change, specify XML encoding mode for the inbound and outbound threads in this type of interface. This must be performed manually. You can use a utility script that looks for translate files that have XML as the input and output format. Then the script can find `NetConfig` routes that invoke these translations. You can then choose what action to take. This type of utility is considered desirable, but not a requirement.

Command-line utilities

The command-line utilities are routines invoked in a shell window. In addition they are used within the IDE for testing, XML complies, and so on.

In all cases, the method of invocation places some restrictions on the use of Unicode for input and output:

- The utilities are implemented in C and take their input from the parameters passed in the locale ANSI codepage. In most cases, you cannot enter these data in UTF-8. Instead, the routines internally change the encoding of those parameters that must be in UTF-8 as required. For example, the XML root element.
- When message-type data files are opened, the file encoding must be UTF-8 to simulate exactly what would happen during engine processing. If the file is in another encoding, then use the `-e` switch to change the encoding before being processed. `-x` is used for `hcitptest` and `hciroutetest`. This corresponds to using the thread inbound encoding selection.
- The utilities currently write their output to `StdOut` or `StdErr` in UTF-8. Because the shell windows, such as `cmd` in Windows, might not display data in UTF-8, it might display UTF-8 data as trash. In this case, pipe the output of the routine to a file and view it in an editor that displays the encoding.
- In the case of Redhat Linux, the locale may be UTF-8 which permits display of message data with that encoding.
- If the environment does not support characters foreign to the locale, then foreign characters cannot be entered and might be displayed as question marks or garbage. For example, character-based terminals.
- In Windows, with the main locale set to English, and the files and other data are in a non-English locale, you must set the non-Unicode applications locale to that of the data, file names, and so on.

hcicmd usage

A message encoding option permits you to specify the encoding for the data file. If there is no encoding specified, then the API keeps the old behavior.

```
hcicmd -p <process_name> -c ". resend_errordb [-m
  <meta_file_name>] <midrange> [<file_name>
  <len10|nl|eof> [-e <msg_encoding>]]"
```

If you only specify a MID to resend, then it is unnecessary to specify the encoding. The engine uses UTF-8 to do the resend.

Restrictions

Variant name, configurator name, and so on, for these might be restricted to ASCII depending on the ability of advanced security to handle foreign content:

- `hcihrltest`
- `hcincdpctest`
- `hciparsetest`
- `hciprocedit`

- `hciroutetest`
- `hciscriptedit`
- `hcitcl,wish,wishx`
- `hcitptest`
- `hcivrltest`
- `hcix12test`
- `hcixlttest`
- `hcixmltest`
- `hcidbdump`

hcitcl

The system sets Tcl system encoding to UTF-8, so the default file encoding is UTF-8.

The default encoding of `StdIn`, `StdOut`, and `StdErr` is ANSI.

hciwish

This is a UTF-8 Tcl command tool providing a window for user interaction and, on Windows, multi-language support (IME on Windows). The system encoding is UTF-8.

The default encoding of `StdIn`, `StdOut`, and `StdErr` is UTF-8.

cvtnull option

When you use the `-cvtnull` option in Tcl message commands, all of the characters following `-cvtnull` are accepted in the system.

Replacing the null with a single character is not reasonable in Unicode.

The reason is that in different languages a character uses a different number of code points. In Arabic it might take five code points to make a character and in French it might take only two code points.

Test steps

```
D:\cloverleaf\cis6.1\integrator\t-tcl.msg\test>hcitcl
hcitcl>set msgvar [msgcreate]
message0
hcitcl>msgset $msgvar "message data"
hcitcl>msgappend -cvtnull ^N $msgvar "^APPENDED ST^UFF"
hcitcl>puts [msgget -cvtnull ! $msgvar]
message data^APPENDED ST^UFF
hcitcl>msgset -cvtnull ^N $msgvar " message data^NAPPENDED ST^NUFF "
```

```

hctcl>puts [msgget -cvtnull ! $msgvar]
message data!APPENDED ST!UFF
hctcl>
D:\cloverrel_58\rats\suites>hctcl
hctcl>set msgvar [msgcreate]
message0
hctcl>msgset $msgvar "message data"
hctcl>msgappend -cvtnull ^N $msgvar "^APPENDED ST^UFF"
hctcl>puts [msgget -cvtnull ! $msgvar]
message data!APPENDED ST!UFF

```

Null characters in messages

UTF-8 Tcl, such as the Unicode standard, avoids the use of the zero-byte null but goes outside of the Unicode standard. It does this with the two-byte sequence 192, 128 to represent a null embedded in the UTF-8 string of the Tcl object. The null byte, with value zero, is reserved for string termination.

In the current byte mode, a message string can contain a null character as a non-terminator, and it can be manipulated using `hctcl` commands. The UTF-8 implementation retains the null character as a zero-byte in the message string. This causes issues for user Tcl which uses the two-byte representation when the single-byte zero is required, retaining the limitations of string terminations in message strings.

Relationship between encoding conversion and UPoCs

The inbound encoding conversion happens before the message is passed to the inbound TPS. Outbound encoding conversion happens after the prewrite UPoC is called. The message data that is passed to all UPoCs are Unicode strings except as noted. The metadata does not get encoding conversion so metadata is always Unicode strings.

UPoC protocol read and write procedures communicate directly with the outside so they deal with encoded byte arrays. Encoding conversion for a UPoC protocol read happens after the read procedure passes the message to the engine. UPoC read and UPoC write are the only procedures that deal with encoded data. All others deal with Unicode strings.

Relationship between encoding conversion, the recovery database, SMAT, and resend

Conceptually, inbound encoding conversion happens after a message is pulled from the inbound pre-TPS queue and before it is passed to the inbound TPS. Outbound encoding conversion happens after a message is returned from the prewrite procedure.

Inbound SMAT is written immediately when a message is received from a protocol driver. Outbound SMAT is written from a copy of the post-TPS queue message after a successful driver write and before the SendOK

procedure is called. Inbound SMAT files contain raw data received from the outside. The inbound encoding conversion has not been performed.

Outbound SMAT files contain UTF-8 Unicode data. The outbound encoding conversion has not been performed.

Recovery database state 1 On IB pre-TPS queue contains raw data from the outside. All other recovery database states contain UTF-8 Unicode data.

Error database state 416 (Inbound Encoding Conversion error) contains raw encoded data. All other error database states contain Unicode.

Messages can be resent from SMAT files or from any arbitrary file using NetMonitor resend. The resend places the message in IB pre-TPS, IB post-TPS, OB pre-TPS or OB post-TPS under user control. Messages resent to IB pre-TPS are considered raw data. The normal inbound encoding conversion happens when the message is pulled from the IB pre-TPS queue. Messages resent to the other three queues are considered UTF-8 Unicode.

This is a consistent treatment of the data. Inbound SMAT is usually resent to IB pre-TPS and outbound SMAT is usually resent to OB post-TPS. The other resend points are only used in special cases.

It may seem inconsistent for inbound SMAT and recovery state 1 to be raw data but there is a good reason for this. Inbound encoding conversion errors can make it impossible to convert the inbound data to Unicode. This is required for inbound SMAT and recovery state 1 to be raw data. Messages with encoding errors can then be examined, corrected and resent in a consistent fashion.

Outbound encoding conversion is implemented as an automatic pre-write.

Note: Outbound encoding conversion is implemented as an automatic pre-write.

Backward compatibility of SMAT files

Unicode stores raw inbound data in SMAT files, but stores UTF-8 Unicode in outbound SMAT files. This causes some backward compatibility issues with existing SMAT files because they contain raw binary data for both inbound and outbound. A single SMAT file could contain raw and UTF-8 data if a site is upgraded with outbound SMAT files in place.

When the Unicode engine is started, it continues writing to the existing SMAT files using UTF-8. This results in mixed raw/UTF-8 files. To handle this properly, the Unicode engine inserts an additional setting into the outbound `.idx` file. This setting is `{ENCODING UTF-8}`.

These settings are stored on a per-message basis, so there is no problem even with a mixed file. When a SMAT file is viewed or resent, the `ENCODING` setting must be checked. If it does not exist, then the message data is assumed to be raw. An automatic binary to UTF-8 conversion is then required when the message is resent to any point other than Inbound pre-TPS.

An automatic UTF-8 to binary conversion is required for any UTF-8 message resent to Inbound pre-TPS. The only valid value for `ENCODING` is UTF-8 and it is only saved for an outbound SMAT file. This feature ensures that both old and new SMAT files can be used without any requirement for explicitly converting the files.

Fixed encoding

Fixed encoding specifies a fixed Big5 encoding. All inbound messages are converted from Big5 to Unicode and all outbound messages are converted from Unicode to Big5.

Invalid inbound byte streams go to the error database. Therefore, a message that starts with A5h, 20h is an error because this is not a valid Big5 byte sequence.

Fixed encoding with "Use byte order mark"

This uses a byte order mark in the inbound message to determine the encoding.

- Byte Order Mark: FE FF
Endian Order: Big
Encoding: UTF-16BE
- Byte Order Mark: FF FE
Endian Order: Little
Encoding: UTF-16LE
- Byte Order Mark: EF BB BF
Endian Order: N/A
Encoding: UTF-8

Messages that start with one of these byte order marks automatically use the associated encoding.

The byte order mark is not required. Any message that does not start with a byte order mark falls back to using UTF-8 encoding from the **Encoding** field. With these settings, the outbound encoding is always UTF-8 and outbound messages do not have a byte order mark.

XML encoding

This uses XML declarations to determine encoding for both inbound and outbound messages. Selecting **XML Encoding** on the Network Configurator causes the **Encoding** field to be unavailable and no input is accepted there.

XML encoding looks for an XML declaration in the message to determine the encoding dynamically. Therefore, a message that starts with this uses Big5 encoding: `<?xml version="1.0" encoding="BIG5" ?>`

XML encoding determination is used both for inbound and outbound messages.

Note that **Use byte order mark** is not selected and messages without an XML declaration are assumed to be UTF-8. Therefore, an inbound message that starts with this uses UTF-8 encoding because the byte order mark is ignored.

The message must start with an XML declaration for it to be recognized.

If **Use byte order mark** had been selected for the previous example, then the leading FE FF would cause this to correctly use UTF-16 Big Endian encoding.

This shows that an XML declaration might not be the most reliable indicator of encoding. An XML message may undergo encoding conversions for storage or transport with the XML declaration being changed. In this case, the byte order mark is considered more reliable than the XML declaration. The XML declaration (BIG5) would be considered the original encoding when the current encoding is UTF-16BE.

XML encoding with "Generate byte order mark"

Generate byte order mark on the Outbound pane of the **Properties** tab uses XML declarations to determine the encoding and generates outbound byte-order marks when appropriate.

Therefore, an outbound message that starts with `<?xml version="1.0" encoding="UTF-32BE" ?>` is output as:

XML processing behavior change

In previous versions, the system did a dynamic encoding conversion during XML-to-XML translation if the encoding was changed by modifying `?xml.&encoding`. This conversion no longer happens because all is internally normalized to Unicode.

Upgraded protocol threads default to binary encoding so that there is no conversion in the threads. This can result in different behavior for sites that use an XML-to-XML translation where the encoding is altered in translation.

You can compensate for this behavior change by specifying XML encoding mode for the inbound and outbound threads in this type of interface. This must be performed manually. You can use a utility script that looks for translate files that have XML as the input and output format. It can then find NetConfig routes that invoke these translations. You could then choose what action to take. This type of utility is considered desirable, but not a requirement.

Example: XML Cyrillic ISO8859-5 to UTF-8 translation

This example shows the operation of the system. An XML-to-XML translation performs a bulkcopy of the content and then copies a literal UTF-8 into the XML encoding declaration.

The input file is in ISO8859-5, a single-byte Cyrillic codepage. In this example, the XML declaration changes from ISO8859-5 to UTF-8; the encoding also changes to UTF-8.

Translate file:

```
prologue
  xlt_infile:    xml text test
  who:
  date:
```

```

xlt_outfile:
  type:      xlt
end_prologue
{ { OP COMMENT }
  { COMMENT {This translation translates the encoding from whatever it is into utf-8 by saving
    utf-8 in the outbound declaration} }
}
{ { OP BULKCOPY }
  { ERR 0 }
}
{ { OP COPY }
  { ERR 0 }
  { IN =utf-8 }
  { OUT ?xml.&encoding }
}

```

Example: XML Cyrillic ISO8859-5 to UTF-8 translation in the system Unicode

Consider [Example: XML Cyrillic ISO8859-5 to UTF-8 translation](#) in the system Unicode. The data is in Unicode before and after the translation so that the encoding conversion makes no sense and does not happen. Everything else is the same. The data is bulk copied and the encoding declaration is changed to UTF-8.

If you convert a site using this translation to the system Unicode, then both threads default to binary encoding. This does not change the data, so that the output message contains ISO8859-5 encoded data with UTF-8 in the XML declaration.

There is another problem with using this site in the system Unicode. The default binary encoding does not change the inbound data. It translates byte values 00h to FFh into the Unicode zero page (U+0000 to U+00FF).

Therefore, the inbound message contains ISO8859-5 values. The XML parser interprets this data as Unicode code points and essentially sees garbage. This Unicode-ized byte data could result in characters that are invalid for an XML file. The translation would then send the message to the error database.

An example of this type of failure would be an input file in UTF-16BE. If the XML file starts with an XML declaration `<?xml ...`, then the inbound byte data (in UTF-16BE) begins with 00 3C 00 3F 00 78 6D 6C. The default binary encoding converts these eight bytes into eight Unicode characters. The XML parser rejects this because 00 3C is not a valid beginning for an XML message.

Both of these problems can be corrected by selecting **XML Encoding** for the inbound and outbound threads on the Network Configurator. This was the primary motivation for adding the XML encoding mode to the protocol threads.

When XML encoding is used for this example, the inbound ISO8859-5 encoding is extracted from the XML declaration. The message is then properly converted to Unicode. The XML parser treats the data correctly and the translation changes the encoding declaration to UTF-8. The outbound thread extracts the UTF-8 encoding from the message data and correctly converts it. This gives the same result as previous versions.

The system cannot automatically detect which threads should or should not have XML encoding enabled, so you must consider this and make the changes. A utility is available that checks for XML-to-XML translations. Threads can also be identified that use these translations or are their destination.

For example, they can be identified in the prologue:

```
prologue
  xlt_infile: xml text test
  who:
  date:
  xlt_outfile: xml text test
  type: xlt
  version: 6.0
end_prologue
```

Configuration tips

Because Java is configured to use UTF-8 by default, the encoding is specified as ANSI when invoking `exec`.

- If trash characters or question marks are displayed for non-English characters, then generally there is an encoding mismatch.
- If the codepage does not contain the character, then it displays as a question mark. For example, Chinese Big5 characters on a U.S. workstation.
- Beware of cp437 on Windows as it is the DOS ASCII/Graphics codepage and some of the graphics glyphs are similar to standard symbols. cp437 is the default codepage for the Command window on many Windows locales.
- In the system, the Java IDE on Windows only reads and writes in the non-Unicode locale.
- Tcl defaults to UTF-8 for file operations and is used in the engine and testing routines to read and write configuration files. That is, there is a mismatch between the IDE and Tcl.
- If using any of the standard I/O channels in `hcitcl`, `hciwish`, or for debugging a procedure, then check the encoding to ensure it is what you expect.
- Use caution with editors, for example, Notepad, that save files in UTF-8 but put the Byte Order Mark at the front of the file in the first three bytes. It might be necessary to remove these using a binary editor if the file is read by the engine or testing routines.
- Note that Tcl has two string-length functions. One returns the character length and the other the byte length of the string. They are not the same.
- Tcl strings are not C strings. Tcl encodes a null character, byte with a value of zero, into a two-byte UTF-8 encoded Unicode code point. Because of this, `\x0`, or any form with or without quotes, is not an empty string. Instead, it is a string with character length one and byte length two. Use `""` for an empty string, quotes without any content, in commands such as `msgget -cvtnull`.
- Configuration files, for example, NetConfig, should use the UTF-8 encoding, not the ANSI locale codepage. This is necessary when the engine employs Tcl to input and decode the contents of the file.

The IDE is implemented in Java, which invokes command-line utilities by `exec` for many of its testing and file functions.

The command-line parameters must be passed in the locale ANSI codepage.

Because Java is configured to use UTF-8 by default, the encoding is specified as ANSI when invoking `exec`.

Configuration of language options in Windows

The system supports all message data using multi-byte characters. To display/input/output the file name, configure the **Regional Language** option to support the language.

To install a localized Windows operating system and use the default setting, you can install the language's operating system and use as-is. The regional/locale/language settings remain the same.

Configuring the regional language setting in English operating systems

If file names and data containing non-English characters are to be displayed correctly, then the OS must be set to the correct locale. If this is not performed, then question marks or unknown characters are in the output.

For example, to configure English Windows so that displays data correctly in traditional Chinese, you must:

- 1 Install the Chinese Font files for East Asian Language.
Alternatively, navigate to any Internet site with Chinese Characters. Internet Explorer prompts you to install that language.
- 2 Configure the regional/locale/language settings that are accessed by the Control Panel's Region and Language option by clicking **Change Keyboards**.
This enables input in Chinese and sets the language for non-Unicode programs.

Configuring the Windows multi-language edition

Windows Multilingual User Interface (MUI) Packs permit you to install an English version of Windows, and then install various User Interface Language Packs. These provide a localized user interface (UI) for the operating system.

For example, you can install a Japanese User Interface Language Pack on top of an English version of Windows. You can then switch the Windows UI language between Japanese and English. The MUI permits you to have multiple language versions of Windows on a single computer.

You can use the Windows MIU to change the user interface, for example, menus and dialog boxes, into another language. Up to 33 different languages can be installed on the machine by an administrator, any user with administrative privileges, using the `muisetup.exe` program.

Changing the user interface language has the effect of displaying menus, dialog boxes and Help files in the specified language.

Note: Changes to the user interface language only becomes effective after logging off and logging on again.

In general:

- 1 Select any one of the installed user interface languages by opening the Control Panel's **Region and Language** dialog box.

- 2 Depending on your OS, navigate to the section where you choose a language for the menus and dialog boxes.
- 3 Navigate to where you can select default user account settings to set the UI language. This is the language that the default user on the local machine, or a domain user that does not have a profile, sees at first log in. This is also the UI language that is applied to system services running on the local machine.
- 4 After changing the UI, go through the steps mentioned in [Configuring the regional language setting in English operating systems](#).

Configuration of language options in Linux

Redhat provides multi-language support. This is installed during the installation. You can then change the default language in the system settings.

You can also change the language setting on the log-in screen. To enable the input method editor (IME), use **Ctrl-Space** or **Ctrl-Shift**.

Redhat Linux installation with only English installed

There are situations where you already have a running system that only uses English. Additional languages are installed during installation.

To manually install other languages:

- 1 Manually install the "kde rpm" from your Redhat CDs. For example, use **kde-i18n-Chinese-Big5XXXXX** on your CD for Traditional Chinese.
- 2 Export/set the language environment variables if they are in English. Use **locale -a** to get a list of locales. Reference from isdred003:

```
[hci@isdred003 ~]$ locale
LANG=zh_TW.UTF-8
LC_CTYPE="zh_TW.UTF-8"
LC_NUMERIC="zh_TW.UTF-8"
LC_TIME="zh_TW.UTF-8"
LC_COLLATE="zh_TW.UTF-8"
LC_MONETARY="zh_TW.UTF-8"
LC_MESSAGES="zh_TW.UTF-8"
LC_PAPER="zh_TW.UTF-8"
LC_NAME="zh_TW.UTF-8"
LC_ADDRESS="zh_TW.UTF-8"
LC_TELEPHONE="zh_TW.UTF-8"
LC_MEASUREMENT="zh_TW.UTF-8"
LC_IDENTIFICATION="zh_TW.UTF-8"
LC_ALL=
```

- 3 Modify `/etc/sysconfig/i18n`.

Reference from isdred003:

```
[hci@isdred003 ~]$ cat /etc/sysconfig/i18n
LANG="zh_TW.UTF-8"
SYSFONT="lat0-sun16"
```

You must install the multi-byte version before doing the changes above. If you have performed the changes before installing the system, then configure a command-line window before launching the system installer, using one of these:

- **setenv LANG en_US.UTF-8**
- **export LANG=en_US.UTF-8**

Encoding conversion between the IDE and the engine

In the engine, data is stored and processed internally in UTF-8. Command-line C programs take ANSI parameters, but they handle data in UTF-8. For ANSI data files, you can use the `-e` switch to specify the encoding, whereupon the system converts that to UTF-8 for internal process. Because it opens/closes a file, the file names must be in ANSI (Windows API only takes ANSI or UTF-16).

Each thread has an encoding which controls the conversion between the external format and UTF-8. An XML encoding mode permits the external encoding to be determined from the XML encoding declaration.

StdIn and StdOut are used when the IDE passes the command-line parameters to the engine or when the engine outputs data to the console/IDE. In the system, the engine always outputs UTF8 data to StdOut if the data originates from a message being processed. The console codepage on Windows is the ANSI codepage, but it can be UTF8 on Linux. The data file can be in a non-UTF8 encoding. This is controlled by the `-e` switch.

Tcl UPoC

The system sets the Tcl system encoding as UTF-8, so the file input and output are assumed to be encoded in UTF-8.

Tcl assumes the file names are UTF-8 if passed by a command-line routine as a string. This is the opposite of the C command-line convention.

The default Tcl StdIn/StdOut/StdErr have their encoding set to UTF-8.

Tcl does not automatically handle the Unicode Byte Order Mark (BOM).

Java UPoC

The JVM of Java UPoC is launched by the Tcl extension framework. Therefore, the file encoding is ANSI; you can work around this issue if you do not use `StdIn/StdOut` and instead pass the strings through JNI. The encoding conversion is not required in this case.

The default encoding of `StdIn`, `StdOut`, and `StdErr` is ANSI, as it does not use system settings in `client.ini`.

UPoC protocol

In Network Configurator, you can select inbound message encoding, for example, Big5. The system engine converts inbound data from Big5 to UTF-8 before pre-SMS (TPS Inbound data). The Read TPS in the **UPoC Protocol Properties** dialog box is called before engine encoding conversion.

The Read TPS has to handle raw data and convert all of the messages into Big5. This is because the engine thinks the inbound data is Big5. In the Traditional Chinese environment, the Read TPS reads Big5 data and sends Big5 data to the engine.

The outbound **Write TPS** of the UPoC protocol is called after thread encoding translation. It is suggested that you remove all string processing from the protocol UPoCs.

UPoCs

A user point of control (UPoC) is a code fragment that directly controls message actions.

UPoCs can modify or control the content and flow of messages, and they can control the start time of messages.

UPoCs also act on issues in protocol, presentation, and translation areas.

Note: For security, best practice is to scan any UPoC code before deployment. See [Security Audit tool](#) on page 1134.

The threads within the system that use UPoCs are:

- Inbound data (protocol issues)
- Pre-translation (presentation and translation issues)
- Outbound data (protocol issues)

UPoC code fragments can be implemented in Tcl or Java. Tcl is the inherent coding interface with the engine, where the engine directly invokes UPoC code segments as Tcl procedures.

Java is an extension of the Tcl interface through the Java Native Interface (JNI).

Because the UPoCs framework is implemented through Tcl, you must understand Tcl and the engine's architecture to understand UPoCs.

A system process is the binary program that is also called the engine. Each process contains command, translation, and protocol threads.

There is exactly one command thread and one translation thread per process, but there can be more than one protocol thread.

There is one Tcl interpreter per thread. A Tcl interpreter is a memory structure that is allocated per thread.

UPoC code that is written in Tcl runs in the same thread as the calling procedure.

Variables follow the normal scoping rules of Tcl. The scope of global variables is the lifetime of the interpreter, which lasts for the lifetime of the thread.

For example, this code makes use of this property:

```
set globalVar 0
proc countMessages {} {
    upvar xlateId          xlateId
    xlateInList             xlateInList
    xlateInTypes            xlateInTypes
    xlateInVals             xlateInVals
    xlateOutList            xlateOutList
    xlateOutTypes           xlateOutTypes
    xlateOutVals            xlateOutVals
    global globalVar
}
```

```
    puts "globalVar: $globalVar"
    incr globalVar
}
```

When this is set as the inbound data TPS UPoC procedure in an inbound thread, `globalVar` becomes the running count of messages that have passed through the current thread.

Scripting languages

Scripting languages are used during interface development. Cloverleaf natively supports multiple scripting languages for interface development, testing, and deployment.

The Cloverleaf integration engine supports these scripting languages as UPoCs for interface development:

- Tcl (natively-embedded)
- Java (natively-embedded)
- Javascript (run through the Java runtime)
- Python (run through the Java runtime)

These are installed and enabled by default, and can be edited in the Script Editor.

The java interface required to run JS and Python scripts is automatically configured when you select JavaScript or Python.

JavaScript and Python must be configured through the Java interfaces. See [Script UPoC](#).

Tcl and Java are supported in the GUI tools so you can select your preferred language. The GUI then loads the correct list of procs.

User-defined scripts are stored in language specific directories at the site and root levels. The master site is respected in the search path.

- `/tclprocs`
- `/java_uccs`
- `/scripts`

Tcl is the highest performing scripting language as it is directly embedded in the C runtime. Java runs in a JVM so is slightly slower than Tcl. JavaScript and Python are extensions to the Java runtime and perform slower than Java.

Usage example

An Interface Developer requires filtering transactions in the Inbound TPS.

Python is used in this instance. The Python directory is located under the `root/scripts` directory. The Developer can then copy and rename the TPS template to `%sitedir/scripts/A05_filter.py`.

In a text editor, the Developer adds the filter code to `A05_filter.py`.

The `A05_filter.py` script is added to the inbound TPS as a Java UPoC on the thread in the Network Configurator.

TPS module interface

The TPS (Tcl Procedure Stream) UPoC is a user-extendable method to control or modify messages.

In some environments, the system must process messages early or late in the message flow, on the inbound or outbound side of a connection. For example, when coalescing messages.

TPS UPoCs are used inside the protocol and translation threads.

In a TPS, a message enters the module at the beginning of the stream. That module can modify, hold, or discard the message.

Any resulting messages are passed along the stream to the next module in sequence.

When the last module in the stream produces a message, the engine considers the TPS processing to be complete and continues processing the message.

Note: Each module in the stream is unaware of its relative position within the stream. A module is also unaware of the function or existence of modules on both sides of it.

Keyed lists

All TPS procedures are invoked with a standard set of arguments. This table shows the arguments that are passed in a keyed list:

Arguments	Description
ARGS	<p>User-supplied arguments, one set for each process, are specified in the Args field of the TPS Properties dialog box.</p> <p>Example:</p> <p>If Args is specified as <code>{client_id test}</code>, then you can use this Tcl code to retrieve it:</p> <pre>keylget args ARGS user_args keylget args ARGS.client_id client_id The value of 'user_args' is '{client_id stop_test}' The value of 'client_id' is 'stop_test'</pre>
CONTEXT	The specific UPoC from which this procedure was called.

Arguments	Description
MODE	<p>The binary mode.</p> <ul style="list-style-type: none"> <code>start</code> is the initial invocation at engine start-up. <code>run</code> is the normal invocation at run time. <code>time</code> is timer-based processing. Used only in UPoC read TPS, HTTP time driven query, and batching time-mode TPS. <code>shutdown</code> runs when a thread is shut down, except in a panic.
MSGID	Message handle associated with the data to be processed.

Contexts

This table shows the UPoC contexts. These are the specific UPoC from which a procedure is called.

Context	Description
ack_control	Protocol ACK/NAK control.
fileset_ibdel	If the Fileset FTP driver is set to local-tps or FTP, then this context is available to override the protocol driver's default setting. The oldest is deleted first. It also programmatically controls which files are deleted after processing.
fileset_ibdirpars	If the Fileset FTP driver is set to local-tps or FTP, then this context is available to override the protocol driver's default setting, oldest read first. It also programmatically controls when files are read in by the driver.
pduppoc_read	Instructions that are provided for the UPoC driver when it performs a read.
pduppoc_write	Instructions that are provided for the UPoC driver when it performs a write.
prewrite	Instructions that are provided for the UPoC driver before it performs the write.
proto_startup	Protocol startup.
proto_statup_sendfail	Notification that is invoked upon an unsuccessful protocol startup.
proto_startup_sendok	Notification that is invoked upon a successful protocol startup.
reply_gen	Reply generation that is invoked after reply time-out.
send_data_ok	Notification that is invoked upon successful data message send.

Context	Description
send_data_fail	Notification that is invoked upon failed data message send.
send_reply_ok	Notification that is invoked upon successful reply message send.
send_reply_fail	Notification that is invoked upon failed reply message send.
sms_fwd_data	TPS data message forwarding stack.
sms_fwd_reply	TPS reply message forwarding stack.
sms_ib_data	TPS IB (inbound) data message stack.
sms_ob_data	TPS OB (outbound) data message stack.
sms_ib_reply	TPS IB reply message stack.
sms_ob_reply	TPS OB reply message stack.
trixd	Transaction ID that is used for translation and routing.
xlt_gen	Generate procedures for route detail.
xlt_post	Xlate post-processing.
xlt_pre	Xlate pre-processing.
xlt_raw	Xlate raw processing.

Message dispositions

Each UPoC module must return a keyed list that instructs the TPS how to process specified messages. These lists are called message dispositions, and they are TPS context sensitive.

This table shows the supported dispositions:

Disposition	Description
CONTINUE	<p>Returns for normal processing and gives to next module in stream. It can continue to:</p> <ul style="list-style-type: none"> • <code>trxid</code>, where a transaction ID is returned. • <code>fileset_ibdirpars</code>, where an ordered list of files to be processed is returned. • <code>fileset_ibdel</code>, where an ordered list of files to be deleted is returned. <p>Otherwise, a message handle is returned.</p>
ERROR	Places the message into the error database and removes it from the recovery database, if required.
KILL	Ends message processing. (<code>msgdestroy</code>)
KILLREPLY	Similar to <code>KILL</code> , but stops waiting for a reply message from the connection. This disposition should only be used in <code>sms_ib_reply</code> .

Disposition	Description
OVER	<p>Instructs the engine to place the message in the queue for the opposite direction. For example, an <code>OVER</code> message in the Inbound Data TPS is given to the OB Data TPS. Only valid in an inbound or outbound TPS.</p> <p>Do not use the <code>OVER</code> disposition within the read and write UPoCs in the UPoC driver. The engine reports an error when it gets this disposition in the read and write UPoCs.</p> <p>Do not use the <code>OVER</code> disposition within the Fileset FTP driver, where it destroys the message. The engine reports an error when it gets this disposition.</p>
PROTO	Places the message at the head of the Outbound Post-TPS Queue (that is, prepared for immediate delivery using the protocol driver). Use <code>PROTO</code> only in a TPS operating within a protocol thread.
SEND	Skips further TPS processing. Sends the indicated messages through the translation thread to their destination.
REROUTE	Chooses a different translation ID route after some translations of a message have taken place. This is used during translation, routing, and in the Route Detail Pre-/Post-Processor.

UPoCs used within the TPS module interface

These UPoCs include:

UPoC	Usage
Protocol Read	This delivers a single inbound data protocol message to the engine from the protocol driver.
Protocol Start-Up	<p>This is the first UPoC that is applied to an inbound data message that is delivered from a protocol driver to the engine. This UPoC controls what happens when the first message is sent to or received from a connection.</p> <p>Actions are:</p> <ul style="list-style-type: none"> • Pass the message to the pre-TPS queue for further processing. • Stop the message. • Generate an outbound protocol message that is delivered immediately after the procedure returns. <p>This UPoC customizes the startup actions of a protocol driver without changing the driver code itself. It is not called again until the thread restarts.</p>

UPoC

Usage

Protocol Start-Up Switch

This is used to control how protocol messages received from the protocol driver are routed. At thread start-up, a Tcl procedure, if defined, is given control of message flow. It is called once to set up its initial state. It is then called each time a new protocol message arrives from the connection. The procedure is given the message and it has the option to do whatever is required.

Each time the procedure is invoked, including the initial invocation, the return value from the procedure is a list of binary object handles wrapped inside two keyed lists.

- The first list indicates binary objects that should be written to the connection.
- The second list indicates binary objects that should be moved to the inbound pre-SMS queue. The second list gives the Tcl procedure the opportunity to pass along protocol messages that it receives that are uninteresting to it.

When the procedure is running, it has access to the inbound pre-SMS stack. It may remove messages from the queue using Tcl built-in functions. This functionality is provided so it can remove messages that have not yet entered inbound SMS processing.

When the procedure is finished controlling the start-up flow, it changes the protocol switch from within Tcl. When the procedure returns after it has thrown the switch, the procedure is not called again until the thread is restarted.

The protocol start-up switch is thrown using the engine-only Tcl command `engpswthrow`. This is required for normal processing. Otherwise, the thread stays in Startup mode.

UPoC	Usage
Inbound Data TPS Processing	<p>An Inbound Data TPS modifies the inbound data messages and creates application-level acknowledgments. Data messages read from a connection are processed through this stream.</p> <p>Messages received from the protocol thread and put into the Inbound pre-TPS queue are saved into Raima and SMAT, if SMAT is defined. At this point, the message is going to the engine, because this message can be recovered /resent from Raima and SMAT.</p> <p>When an Inbound Data TPS is called, messages are placed in the inbound message queue. The input is recently-read data protocol messages. The output is pre-translation data messages. Resulting engine messages are sent to the translation thread for processing.</p> <p>This UPoC is used to:</p> <ul style="list-style-type: none"> • Coalesce messages • Divide messages • Convert from EBCDIC to ASCII • Create acknowledgment messages
Outbound Data TPS Processing	<p>This modifies the outbound message data.</p> <p>This UPoC is used to:</p> <ul style="list-style-type: none"> • Coalesce messages, to create a batch file • Divide messages, when remote connection power is limited • Convert ASCII to EBCDIC <p>When the Outbound Data TPS is called, it modifies the message immediately before it enters the queue to be written to the destination connection. The input is post-translation data messages, and the output is to-be-written outbound data protocol messages.</p> <p>All post-translation data messages are processed through this stream. Resulting messages are placed in the outbound data queue.</p> <p>If message forwarding is enabled, then data messages are run through this UPoC on the forwarded thread.</p>

UPoC**Usage**

Route Detail Pre-Processor

The XLT Route Detail Pre-Processor works on a copy of the original message, and processes the message before the route detail translation actions are run. Each route detail gets its own copy of the resulting message from the pre-processor.

This UPoC is used to:

- Process an entire inbound message before it goes into translation.
- Transliterate operations.
- Modify all translated versions of a message

Route Detail Post-Processor

This processes the message immediately after the detail transaction actions are complete.

This UPoC is used to:

- Modify an entire outbound message.
- Convert an entire message to uppercase.

UPoC

Generate routes

Usage

These affect message routes and translations, where the destination and translation used to process messages are under programmatic control.

Typically, use Generate routes to route a message that is based on its content. These commands are valid:

- `msgrouteget msgId` retrieves the message's route detail list, which is a list of keyed lists from the Network Configurator file.
- `msgrouteset msgId list` replaces the message's routing list.

If you do not use `msgrouteset`, then the route detail list retains its original value. These commands work only during Generate route processing.

Because Generate routes affects messages dynamically, there are these considerations:

- In any route detail list, only the first Generate is guaranteed to run. Route details after the Generate are available to the route's procedure, but they might not be run.
- If the route detail list is empty or it cannot be internalized, then the original source message is moved to the error database. For example, if there is bad format or typographical errors in the keyed lists.
- Generate procedures can `SEND` and `KILL` messages, but all `CONTINUE` messages must have route detail lists attached to them.

UPoC	Usage
Raw routes	<p data-bbox="818 302 1409 401">These turn a single source message into any resulting messages without applying a translation to the message.</p> <p data-bbox="818 415 1409 445">A Raw route detail is a list of Tcl procedures that are:</p> <ul data-bbox="818 459 1409 567" style="list-style-type: none"> • Run in sequence • Called with the same keyed-list interface • Called with the same expected keyed-list output <p data-bbox="818 581 1409 783">Upon entry to each procedure, the source message's <code>msgXlateDests</code> metadata field contains the list of destination connections to which the message is sent. Before the first procedure is called, the source message's metadata is set to the default list of destinations that are configured for the route detail.</p> <p data-bbox="818 798 1409 896">Each procedure can modify the list of destination connections. If no modifications are made, then the default list is used.</p> <p data-bbox="818 911 1409 974">Modifications to the list can be made by direct manipulation or by calling a programmatic interface.</p> <ul data-bbox="818 989 1409 1339" style="list-style-type: none"> • The first keyed list (messages that should be given to the remaining procedures, if any) is tagged with a key of <code>CONTINUE</code>. • The second keyed list is tagged with a key of <code>SEND</code>. The keyed list contains messages that should be sent to a connection independent from the processing of messages in the other keyed list. • Each message that is listed in the <code>SEND</code> keyed list is immediately placed into the Protocol queue without further processing. <p data-bbox="818 1354 1409 1587">When the translation is complete, all messages in the Partial queue are moved to the correct post-translation queue. This includes messages that are placed there by <code>SEND</code> operations. This movement takes place in a FIFO message priority order. During the movement, <code>SEND</code>-generated messages are not treated specially.</p>

UPoC	Usage
Reply Generation	<p>Reply generation is called only if the engine is expecting an inbound protocol reply message, but does not receive one. This UPoC exists because replies must be received within a configurable time-out period.</p> <p>The input <code>msgid</code> is a COPY of the outbound data message that generated no reply. It is meant to be used as a reference for generating the inbound reply.</p> <p>If the input <code>msgid</code> is returned with <code>PROTO</code> disposition, then the message is placed directly on the OB Processing queue. This message is not recoverable. To resend a recoverable message, use the <code>OBMSGID</code>.</p> <p>The <code>OBMSGID</code> is the handle for the saved outbound data message that generated no reply.</p> <p>Typically, use Reply Generation to:</p> <ul style="list-style-type: none"> • Resend the outbound data message. • Receive message handles, zero or more, that are used as inbound reply protocol messages. <p>A source protocol thread waits on a reply for a defined amount of time. If no time-out is configured, then it waits indefinitely.</p>
Pre-Write TPS	<p>Use this for a final chance to edit a particular message before protocol write.</p> <p>The Pre-Write TPS is used in sequence number synchronization and time-based logic.</p> <p>The Pre-Write TPS is configured on the Network Configurator.</p> <p>The only dispositions that are permitted are <code>CONTINUE</code> and <code>KILL</code>.</p> <p>If a protocol write fails, then it can be re-queued to the outbound post-TPS queue for several retries before the write is considered to have failed. <code>SENDFAIL</code> controls additional retries if <code>TRYAGAIN</code> is set.</p> <p>These retries cause the message to flow from the outbound post-TPS queue to the protocol specific write again, therefore going back through the pre-write UPoC. A message flag, <code>MSG_FLAG_DONE_PREWRITE</code>, is set when the message has successfully passed through a pre-write UPoC.</p>

UPoC	Usage
Replies	<p>Reply Message processing uses the reply TPS queue, and the reply message destination determines the message format. The inbound reply message destination is determined by the outbound message source.</p> <p>There are two types of reply message TPS:</p> <ul style="list-style-type: none"> Inbound Reply TPS This processes all reply protocol messages read from a connection. The resulting engine messages are sent to the translation thread. Input is recently-read reply protocol messages. Output is pre-translation reply messages. Outbound Reply TPS This processes all post-translation reply messages. The resulting protocol messages are placed in the outbound reply queue. Input is post-translation reply messages. Output is to-be-written outbound reply protocol messages. If message forwarding is enabled, then reply messages are sent through on the forwarded thread.
Send OK TPS	<p>This UPoC is called only if the outbound protocol message is successfully sent.</p> <p>Typically, use the <code>Send OK TPS</code> to update application databases or to update other operations that are associated with the successful delivery of the message.</p> <p>This UPoC receives a copy of the outbound message as an argument.</p>
Send Fail	<p><code>Failed Data Message Send</code> is a TPS called only if the outbound protocol message is not successfully sent.</p> <p>Typically, the failed data message send:</p> <ul style="list-style-type: none"> Receives a copy of the outbound message as an argument. Produces special message metadata that describes how the message failed.

Code fragment interface

Code fragments are one- or two-line code snippets that are used only in the translation thread UPoCs. They are different from TPS in that users write the code.

Code fragments are evaluated at the global scope after special Tcl global variables are set.

This table shows the included global variables:

Global variable	Description
<code>xlateInVals</code>	Retrieved value strings.
<code>xlateInTypes</code>	Retrieved data types.
<code>xlateOutVals</code>	Value strings going into the next translation phase.
<code>xlateOutTypes</code>	Data types going into the next translation phase.
<code>xlateId</code>	The Translation Pseudo Machine (XPM) handle for use with <code>xpm</code> commands.
<code>xlateInList</code>	The list of input objects.
<code>xlateOutList</code>	The list of destination objects.

Code fragment UPoCs

These code fragment UPoCs work within the Code Fragment Interface.

This table shows the code fragment UPoCs:

UPoC	Description
Xlate Action Pre-Processor	<p>This UPoC processes the source values before the translation action completes.</p> <p>You can use this UPoC to choose a subset of source values based on a source value.</p> <ul style="list-style-type: none"> <code>\$xlateInVals</code> and <code>\$xlateInTypes</code> are the data fetched from the input list. <code>\$xlateOutVals</code> and <code>\$xlateOutTypes</code> are initialized to <code>\$xlateInVals</code> and <code>\$xlateInTypes</code> respectively. <code>\$xlateOutVals</code> and <code>\$xlateOutTypes</code> can be modified to determine what values are included in the main translation phase. <p>An error happens if <code>\$xlateOutVals</code> does not exist at the end of the callout.</p>

UPoC	Description
Xlate Action Body UPoC	<p>This UPoC is available only with the CALL translation action within a translation.</p> <ul style="list-style-type: none"> • <code>\$xlateInVals</code> and <code>\$xlateInTypes</code> are the values entering this phase. • Modifying <code>\$xlateOutVals</code> and <code>\$xlateOutTypes</code> has no effect. • Because XPM does not perform implicit data store, use <code>xpmstore</code>. • <code>\$xlateOutList</code> is for reference only. <p>This UPoC gives ultimate control over the translation, and defines special actions that are not intrinsically available in the translator.</p>
Xlate Action Post-Processor	<p>This UPoC processes the destination values immediately after the translation action completes.</p> <ul style="list-style-type: none"> • <code>\$xlateInVals</code> and <code>\$xlateInTypes</code> are the values of the actions' destination objects after the main body. • There is no automatic storing of message elements. Therefore, modifying <code>\$xlateOutVals</code> and <code>\$xlateOutTypes</code> has no effect.

Transaction ID determination interface

The Transaction ID Determination Proc determines the transaction ID if it cannot be determined from a configured offset and length. You can choose the transaction ID and examine the data inside the message for special cases.

This UPoC does not generally apply to UN/EDIFACT, HL7, or X12 messages. The standards define how the transaction ID is determined.

These types of transaction IDs are available: offset/length, proc, and hard-coded string.

The return value from the Transaction ID Determination UPoC is the transaction ID the engine uses.

Arguments in Transaction ID Determination UPoCs

In Java UPoC TrxID determination, `Process(cloverEnv, msg)` has been deprecated.

`userargs` is now supported. For example, `Process(cloverEnv, msg, userargs)`.

When no `userargs` are specified from the NetConfig file, `userargs` is null.

There are two ways to reuse the existing `Process(cloverEnv, msg)`:

- Invoke it in a new `Process(cloverEnv, msg, userargs)`:

```
public class Trixid extends Trxid {
    public String process (CloverEnv cloverEnv, Message msg, PropertyTree userargs)
    throws CloverleafException {
        if (userargs == null)
            return process(cloverEnv, msg); // re-use the existing Trxid JavaUPoC here.
        return null;
    }

    public String process (CloverEnv cloverEnv, Message msg)
    throws CloverleafException {
        :
        // the existing logic for trxid in JavaUPoC
        :
    }
}
```

- Fill `Process(cloverEnv, msg, userargs)` with the existing logic of `Process(cloverEnv, msg)`.

Guidelines

There are several UPOC usage guidelines to help decide which UPoC to use, where to put code, and how to write the code.

Write an inbound or outbound TPS module if you:

- Do not require the protection of the Partial queue
- Are dealing with protocol-level issues
- Require to use the PROTO disposition
- Require inbound data to affect inbound or outbound replies in some way
- Require interstream communication

Write a prewrite module if you:

- Require any session-specific data in the message
- Require any timing-specific data in the message
- Require any session or timing specific data as the message is handled

Write a route detail module if you:

- Require the protection and recoverability of the Partial queue
- Are dealing with translation/presentation issues

Write an action pre- or post-processor module if you:

- Require modifying a source or destination value in a single translation action
- Require initiating a difficult presentation action on a single action's resulting values

Use GENERATE routes to solve translation and routing issues, as required.

Write Send OK/Send Fail modules to take some action, such as operator notification, after the attempted delivery of an outbound message.

Using HAPI

Before using HAPI (HL7 Application Programming Interface) in Java UPoC and Java driver, you must verify if these issues exist. This table shows the possible issues and their workarounds:

Issue	Description	Workaround
HAPI Class conflict with Java UPoC	The Message class that is defined in HAPI has a conflict with Message class in Java UPoC.	Use the HAPI Message class with the full package name, such as <code>ca.uhn.hl7v2.model.Message</code> .
ClassLoader issue	The way <code>ClassLoader</code> is used in HAPI has issues in JavaUPoC and Java Driver.	Configure HAPI <code>ClassLoader</code> in the runtime as required. For example: <pre>ClassLoader cloader = Thread.currentThread().getContextClassLoader(); if(null == cloader) { Thread.currentThread().setContextClassLoader(ClassLoader.getSystemClassLoader()); };</pre>
Class-path setting	By default, HAPI packages are not listed in the JVM classpath in Cloverleaf (<code>\$ROOTPATH/java_uccs/HAPI/lib</code>).	<pre>[jvm] classpath=\$ROOTPATH/java_uccs/HAPI/lib/*</pre>

The HAPI libraries in the CAA-ION BOX are located in `%HCIR00T/java_uccs` when the BOX is deployed.

Although CAA-ION is only available on Windows platforms, these libraries are available on all platforms.

UPoC Debugger solution

UPoC debugging support has not been added to the Cloverleaf IDE. As an interim solution, you can use NetBeans as the user-facing debugger.

Netbeans is the official IDE for Java 8. It has built-in debugging support for Java and JavaScript. Plug-ins for Tcl and Python are also available. The testing tools and server Tcl have been modified so that NetBeans can be used for Cloverleaf UPoC debugging.

A Tcl plug-in is available for NetBeans that works on NetBeans 8. To make it work with UPoC testing tools, changes are required on both the plug-in and the Cloverleaf server.

Debugging protocol

A line-based text protocol is used by the Tcl plug-in. The user sends a request to the server, and the server sends a response back to the user.

The response is optional. For the run requests `stepover`, `continue`, and so on, no response is sent back.

This protocol must be implemented in the Tcl server.

Request and response formats are:

- Status request and response.

Status request is used to get the current location where the program stops.

```
Request:
status
Response:
<file>:<line>
```

<file> is a full path name and <line> is a line number.

- File request and response.

This file request does not exist in the original Tcl plug-in. It is used to retrieve a source file from the server.

```
Request:
file <file>
Response:
<N>
<line 1>
...
<line N>
```

<file> is a full path name, <N> is the number of lines in the file, and lines <line 1>...<line N> are the file contents.

- Variables request and response.

Variables request is used to get names and values of current visible variables, including local variables and global ones.

```
Request:
variables
Response:
<M>
<lvar_1>=<lval_1>
...
<lvar_M>=<lval_M>
<N>
<gvar_1>=<gval_1>
...
<gvar_N>=<gval_N>
```

<M> and <N> are the numbers of local and global variables respectively. <lvar_x>/<gvar_x> is a local/global variable name and <lval_x>/<gval_x> is its value.

This is an example of variables response:

```
2
args="{mytest: calling a proc in this file}"
i="0"
0
```

This means there are 2 local variables `args` and `i`, and no global variable.

- Callstack request and response.

Callstack request is used to get the callstack information of the stopped program.

```
Request:
callstack
Response:
<N>
<frame_1>
...
<frame_N>
```

`<N>` is the number of frames, and `<frame_x>` is the frame information that has four lines with this format:

```
<Tcl command>
<level>
<file>
<line>
```

`<Tcl command>` is the Tcl command string of the frame. `<level>` is the level number. `<file>` is the full path name. `<line>` is the line number.

This is an example of a callstack response:

```
3
"mytest {MSGID message0} {CONTEXT sms_ib_data} {ARGS {}} {MODE run} {VERSION 3.0}"
1
???
1
"proc_in_this_file \"mytest: calling a proc in this file\""
2
/work/co_dev/6.2P/integrator/t-dtc/tclprocs/mytps.tcl
53
"for {set i 0} {$i < 5} {incr i} {\n proc_deep $i\n}"
3
/work/co_dev/6.2P/integrator/t-dtc/tclprocs/mytps.tcl
21
```

In this example, there are three frames total.

Note: If the file name is `???`, then this indicates the command is not from a file. It could be a string being evaluated. The Tcl command can be a multi-line string in which the new line character is replaced with the escape sequence `\n`.

- Breakpoints request.

Breakpoints request is used to set breakpoints. No response.

```
Request:
breakpoints
<N>
<breakpoints_in_file_1>
...
<breakpoints_in_file_N>
```

`<N>` is the number of files with breakpoints set, and `<breakpoints_in_file_x>` is the breakpoint information that has two lines, using this format:

```
<file>
<line_1> <line_2> ... <line_M>
```

<file> is the full path name of a file with breakpoints in it, <line_x> is a line number.

This is an example of the breakpoints request:

```
breakpoints
1
/work/quovadx_dev/qdx6.2P/integrator/t-dtc/tclprocs/mytps.tcl
14 53
```

In this example, the `mytps.tcl` Tcl source file contains two breakpoints set at line 14 and 53.

- Program run requests are used to control the running of the program. There is no response for these requests.

```
Requests:
continue
step
stepover
stepout
```

Request `step` is the same thing as `stepinto`.

Attach Debugger

The original NetBeans Tcl plug-in can only do local debugging. For example, starting a Tcl program and controlling its running.

The plug-in cannot:

- Attach to a running Tcl program.
- Debug a Tcl program running on another machine.
- Debug a Tcl script run by a Tcl interpreter that is embedded in a C program. For example, the Cloverleaf testing tools.

To debug Cloverleaf UPoCs, the Tcl plug-in must be extended with the `attach` and remote debugging functionality. A CLUD/Tcl debugger type is added in the **Attach Debugger** dialog box. Users can specify the target machine, using the host name or IP address, and TCP port.

Local copies of source files

The Tcl source files opened in NetBeans during a debugging session are local copies retrieved from the debugger server. The files are saved in a temporary folder and their contents are shown read-only to prevent any modification.

You must use the Cloverleaf IDE to do Tcl source file editing. Ensure the changes are made to the server files and that `tclIndex` is updated. After this, you can debug again to see the changes take effect in NetBeans.

When the debugging session ends, the temporary folder and all files in it should be deleted.

NetBeans for Java/JavaScript UPoC debugging

NetBeans has built-in support for Java and JavaScript debugging. To debug Cloverleaf Java/JavaScript UPoC, the `CLJAVA_INIT` environment variable must be set to start the JVM in debugging mode.

```
$ setenv CLJAVA_INIT -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=<port>
```

After the testing tool starts, the JVM listens on the TCP port `<port>` and waits for the debugger to attach.

Python UPoC debugging

In the interim solution, Python UPoC debugging is only supported in the command line using Python's `pdb` module. The `{DEBUG 1}` argument is used to enable debugging.

For example:

```
$ hcitptest -r run -x ASCII -f nl -c sms_ib_data -e "hcitptestshowbydisp "
$HCISITEDIR/in.txt 'cljTPS {CLASS ScriptTPS} {LANG python}
{SCRIPT "if mode == \"run\":\n msg.setContent(msg.getContent().upper())\n dl.add(dl.CONTINUE,
msg)"}
{DEBUG 1}'
```

CLDI

The Cloverleaf Debugging Interface (CLDI) defines the interface between Cloverleaf Debugger client and the Tcl/Java Debugger servers. This includes the format of the requests from the client to the server, for example, setting breakpoints. It also includes the information reported in reverse, for example, the stack frames when a breakpoint is hit.

The transport mechanism is a TCP connection between the client and server. When the `hcitptest` TPS testing tool starts, the server is listening on a TCP socket, waiting for the client to connect.

The listening port number can be allocated by the operating system or specified with the `hcitptest` command-line arguments. The file `dbg_port.PID` is created in `$HCISITEDIR/exec/debugger` with the port number as the file contents. `PID` is the process ID of the `hcitptest` process.

The debugging session begins after the TCP connection is established. The listening port is closed, and the `dbg_port` file is deleted.

When the program stops running because of any stop event, the server reports with information on the current situation. The client can send various debugging commands to the server and receive the response information that is sent back. In this way, you can examine a variables' values, single step, or set more breakpoints, then resume running the debugger.

Debugging command syntax

From the client to the server, the debugging command syntax is similar to the normal shell command line. This includes a command word and some arguments:

```
command [arg1 arg2 ...]
```

Debugging information syntax

From the server to the client, the debugging information is notifications reported when events happen, or responses to the debugging commands. Every notification or response is a JSON object with this format:

```
{"status": status-string, name1: value1, name2: value2, ...}
```

The value `status-string` can be one of:

- `connected`
- `started`
- `running`
- `stopped`
- `done`
- `ended`

For example, when a connection is established, a notification similar to this is sent by the server:

```
{"status": "connected", "host": ":::1", "port": "46145"}
```

Magic token (authentication request)

When the user successfully connects to the server, the user's machine XORs the debug token and 1129076036. For example, 0x434c5544, ASCII code of “CLUD,” to get the magic token. It then sends a single line of this magic token to the server as an authentication request. The server checks its correctness. If it is in error, then the connection is immediately closed.

For example, with a debug token 858680469, the magic token is 1885485521 because $858680469 \text{ XOR } 1129076036 = 1885485521$.

Debugging Tcl UPoC with NetBeans

To begin debugging, ensure the NetBeans IDE is installed. See [#unique_1344](#).

After installing the NetBeans IDE, you can start `hcitptest` and run NetBeans.

On NetBeans, select **Debug > Attach debugger**.

In the **Attach** dialog box, select **CLUD-Tcl** from the Debugger list. Then, set the host name and port number in **Target**. For example, **hostname:6688**.

These resources are available:

- [Local copies of Tcl source files](#)
- [Using Step Over and Step Into](#)
- [Variables tab](#)
- [Call Stack tab](#)

Installing the Tcl plug-in and running NetBeans

Ensure the NetBeans IDE is installed. See #unique_1344.

- 1 Start the NetBeans 8.2 IDE.
- 2 On the IDE, select **Tools > Plugins**. This opens the **Plugins** dialog box.
- 3 In the **Plugins** dialog box:
 - a Select the **Downloaded** tab and click **Add Plugins**.
This opens the **Add Plugins** dialog box.
 - b Locate the Tcl plug-in .nbm file. This is located in `$HCIR00T/contrib/NetBeansTcl/org-netbeans-modules-languages-tcl.nbm`.
 - c Click **OK** to close the **Add Plugins** dialog box. Then, click **Install** on the **Plugins** dialog box.
- 4 In the **NetBeans IDE Installer**, click **Next**. Accept the license and click **Finish** to complete the installation.
- 5 Restart NetBeans.
- 6 In a terminal, run `setroot`, then `setsite` to the site.
- 7 Start `hcitptest`:

```
$ hcitptest -P 1234 -r run -a -x ASCII -f nl -c sms_ib_data
-e "hcitptestshowbydisp " $HCISITEDIR/in.txt "mytest"
```

`-P` specifies the listening port.

This results in the message:

```
File cis_site/cis19.1/integrator/t-dtc/exec/debugger/cludport.37481 is created.
Listening on port 1234.
...
```

- 8 Add `-C "tcl_scripts"` to evaluate Tcl scripts before the debugged proc is run.
For example, you can use `-C "source myproc.tcl"` to load another Tcl file `myproc.tcl` in which a proc to debug is defined:

```
$ hcitptest -C "source myproc.tcl" -P 1234 -r run -a -x ASCII -f nl
-c sms_ib_data -e "hcitptestshowbydisp " $HCISITEDIR/in.txt "mytest"
```

- 9 Run NetBeans.
- 10 Select **Debug > Attach Debugger** or click the tool button. This opens the **Attach** dialog box.
- 11 In the dialog box, in **Debugger** select **CLUD - Tcl**.
For **Target**, specify the host name and port number. For example, **hostname:1234**.
File `mytps.tcl` is opened and the program stops at the first line of proc `mytest`. Now you can set breakpoints, step through the code, check call stack, and change variable values.

Local copies of Tcl source files

During a debugging session, the Tcl source files that are opened in NetBeans are local copies that are retrieved from the Debugger engine. They reside in a subfolder in the system's `temp` folder. In Windows, this is `%TEMP%`. In UNIX, this is `/tmp`.

The file name has the format `CLUDtimestamp`. For example, `CLUD20170608002457504`.

These local copies are read-only, and cannot be modified in NetBeans. You must use the Cloverleaf IDE to do Tcl source file editing. This ensures that changes that are made to `tclIndex` and to files on the server are also updated.

Then, you must debug `hcitptest` to see the changes in NetBeans.

When the debugging session ends, the subfolder and all files in it are automatically deleted.

Using Step Over and Step Into

Unlike other programming languages, the `if-else`, `switch`, `for`, and so on, control structures in Tcl are commands instead of special grammar tokens. When they are entered, a new stack frame is created. This means if you use `Step Over` on them, you skip the whole structure. To enter the clauses or bodies of these structures, you must use `Step Into`.

Use `Step Out` to get out of the current `frame`. Normally, this means getting to the caller `proc`. Sometimes, the program stops in an `if-else`, `switch`, and so on, control structure. In these cases, when using `Step Out` it arrives at the next command after the structure.

For more information on `frame`, see the manual page of `info frame`. See <https://www.tcl.tk/man/tcl/TclCmd/info.htm#M15>.

"TIP 280: Add Full Stack Trace Capability With Location Introspection" has additional information. See <https://core.tcl.tk/tips/doc/trunk/tip/280.md>.

Variables tab

All current visible variables are shown in the NetBeans IDE **Variables** tab. This tab has these columns:

- Name
- Type
- Value

To change a variable's value, click and edit the value in the Value column. Then, press **Enter** or click in the Name or Type column to stop editing.

The values that are shown are always enclosed by double quotes. You are not required to enclose your entry with double quotes if there is no white space or double quote character in the field. The `set` command with your input, as given, is sent to the Tcl Debugger engine for evaluation. You must ensure the syntax is correct.

For example, you can change a variable `debug`'s value to `1`, `"1"`, or `{1}`. The results are identical. The commands sent to the server are:

- `set debug 1`
- `set debug "1"`
- `set debug {1}`

If there are white spaces or double quote characters in the value string, then ensure the syntax is correct. For example:

```
set str "hello world"
set str {hello world}
set str hello"world"      <- This is correct
set str {hello"world"}
set str "hello\"world"
```

These commands are incorrect. The Tcl interpreter raises an exception and terminates running when these are used:

- `set str hello world`
- `set str "hello"world`

Call Stack tab

All procs and commands in the current chain of calls are shown in the NetBeans IDE **Call Stack** tab. These columns are on the tab:

- Frame
- Command String
- FileName:LineNumber
- Level

The current command is shown at the top. It has the largest frame number. The callers are shown individually below the top frame.

The program does not stop before the current command. It is in the current command, but before the command is run. This means the current command string is after command substitution and variable substitution. See <http://tcl.tk/man/tcl8.4/TclCmd/Tcl.htm>. This lists a summary of Tcl language syntax.

For example, the value of variable `i` is 5.

- If the command in the source code is `set x $i`, then the Command String column entry is `set x 5`.
- If the command in the source code is `set x [expr $i*2]`, then the Command String column entry is `set x 10`.

Example

```
proc_in_this_file { args } {
    variable a
    #uplevel 1 echo [set args]
    #set str "proc_in_this_file: $args"
    #puts stdout "proc_in_this_file: $args"
    for {set i 0} {$i < 5} {incr i}
        { proc_deep $i
        }
    set a $i }
```

A breakpoint is set in the `for` loop body.

When the program stops here, the current command string is `proc_deep 0`. `$i` is substituted with its value `0`.

Now, even if you change the value of variable `i` to `3`, when you step into `proc_deep`, the argument is still `0` instead of `3`.

When the body run is completed, before and after `incr i` is run, the new value of `i` takes effect, changing from `3` to `4`.

Debugging Java/JavaScript UPoC with NetBeans

- 1 In a terminal, run `setroot`, then `setsite` to the JavaScript/Python demo site.
- 2 Set the `CLJAVA_INIT` environment variable for remote debugging:

```
$ setenv CLJAVA_INIT -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,
address=12581
```

- 3 Start `hcitptest`, using:

```
$ hcitptest -r run -x ASCII -f nl -c sms_ib_data -e "hcitptestshowbydisp"
$HCISITEDIR/in.txt "cljTPS {CLASS ScriptTPS} {LANG javascript} {FILE per.js} {FUNC tpsFunc}"
```

For `hcitptest`, the command line argument depends on the scripting language:

- Python:


```
cljTPS {CLASS ScriptTPS} {LANG python} {FILE per.py} {FUNC tpsFunc}
```
- JavaScript:


```
cljTPS {CLASS ScriptTPS} {LANG javascript} {FILE per.js} {FUNC tpsFunc}
```
- Java:


```
cljTPS {CLASS TestDispList}
```

- 4 Run NetBeans.
- 5 Select **Choose menu File > Open File**. Locate and open the `$HCISITEDIR/java_uccs/JavascriptTPSBeta.java` Java source file and the `$HCISITEDIR/javascriptmodules/simpleUPoC.js` JavaScript source file. Set breakpoints in the files by clicking the left margin of the lines where you require the program to stop.
- 6 In the NetBeans IDE, select **Window > Debugging > Sources**. This opens the **Sources** tab.
- 7 Click the **Remote Attach** tab.
 - a Right-click in the tab and select **Add Source Root**.
 - b Navigate to, and add, these files: `$HCISITEDIR/java_uccs` and `$HCISITEDIR/javascriptmodules`
- 8 In the NetBeans IDE, select **Debug > Attach Debugger**. You can also click the tool button. This opens the **Attach** dialog box.
- 9 Specify the **Host** and **Port**. If Netbeans is running on the same machine as the Cloverleaf server, then set **Host** to `hostname`. Then, click **OK**.

- 10 When `hcitptest` stops at a breakpoint, you can examine the call stack, change variable values, and so on.

Debugging Python UPoC

Python UPoC debugging is supported only from the command line, not with NetBeans.

- 1 In the shell of a terminal, run `setroot`, then `setsite` to the JavaScript/Python demo site.
- 2 To debug Jython TPS, in the `hcitptest` command line, add `{DEBUG 1}` to enable debugging:

```
$ hcitptest -r run -x ASCII -f nl -c sms_ib_data -e "hcitptestshowbydisp" $HCISITEDIR/in.txt
"cljTPS {CLASS ScriptTPS} {LANG python} {FILE per.py} {FUNC tpsFunc}"
The simpleUPoC module is imported
> /work/quovadx_dev/qdx6.2P/integrator/jsjy/pythonmodules/simpleUPoC.py(17)run()
-> '@type msg: Message'
(Pdb)
```

At this point, you can use various commands to do debugging. Specify “h” for help. See the online *pdb - The Python Debugger* document for reference.

<https://docs.python.org/2/library/pdb.html>

- 3 In Emacs (Version 24.3.1 on CentOS 7.1), add these lines to the `~/.emacs` file:

```
(require 'python)
(add-hook 'comint-output-filter-functions 'python-pdbtrack-comint-output-filter-function)
```

- 4 Start Emacs, then run the `hcitptest` command in any comint-based buffers. For example, `*shell*`. The `pdbtrack` hook acknowledges the `pdb` prompt and presents the line in the source file where the program is stopped in a pop-up buffer.

Debugging notifications

Notifications are sent by the server when some events occur.

Connection notification

When the client connects to the server and authentication is successful, a connection notification is reported with the client host IP and port number. For example:

```
{"status": "connected", "host": "192.168.1.104", "port": "42804"}
```

TPS proc start/end notification

When the TPS proc is started or ended, a notification is reported:

```
{"status": "started", "cmd": command_string}
```

or

```
{"status": "ended", "cmd": command_string}
```

The value of `command_string` is the TPS proc command string. For example:

```
{"status": "started", "cmd": "mytest {MSGID message0} {CONTEXT sms_ib_data} {ARGS {}}  
{MODE run} {VERSION 3.0}"}
```

Program stopped notification

When any stop events occur, a program stopped notification is reported with current situation information. The format is:

```
{"status": "stopped", "reason": reason_string, "frame": frame_number, "command":  
command_string, "callstack": array_of_frames, "vars": variables_object}
```

- The value of `reason_string` is the stop reason: begin, breakpoint, or stepping.
- The value of `frame_number` is the number of the current frame.
- The value of `command_string` is the Tcl command to be run. This command string is after command substitution and variable substitution.
- The value of `array_of_frames` is a JSON array of frames. Every frame is a JSON object with this format:

```
{"level": level_number, "frame": frame_number, "type": type_string, "line": line_number,  
"file": file_path, "cmd": command_string, "proc": proc_name}
```

The values of `array_of_frames` are:

- `level_number`, `frame_number`, `line_number`
These are the number of levels, frames, or lines in a file.
- `type_string`, where `type` is source, proc, eval, or precompiled.
- `command_string`
This is the Tcl command being run. This command string is unsubstituted as in the source code.
- `proc_name`
This is the name of the proc that the current command is in.
- The value of `variables_object` is all visible variables in the current frame with this format:

```
{name1: value1, name2: value2, ...}
```

Debugging commands

When `hcitptest` is stopped, the client can send debugging commands to the server and receive responses.

Breakpoint management

Debugging commands for manipulating breakpoints include:

- Insert a breakpoint

```
b file:line[?condition]
```

`file` must be an absolute path (Unix) or fully-qualified path (Windows, beginning with a driver letter and “:”) of the Tcl source file.

`line` is the line number.

The optional `condition` is a Tcl expression. For example:

```
b /work/quovadx_dev/qdx6.2P/integrator/t-dtc/tclprocs/myproc.tcl:12?$i==3
```

If there are whitespaces in the file path or in the Tcl expression, then “” or “{ }” must be used. For example:

```
b "/work/quovadx_dev/qdx6.2P/integrator/t-dtc/tclprocs/myproc.tcl:12?$i == 3"
```

or

```
b {/work/quovadx_dev/qdx6.2P/integrator/t-dtc/tclprocs/myproc.tcl:12?$i == 3}
```

The response to the `b` command is:

```
{"status": "done", "dbgcmd": "break"}
```

- Delete an existing breakpoint

```
d file:line[?condition]
```

The argument must be exactly the same as given in a previous “`b`” command. Otherwise, the breakpoint is not deleted.

The response to the `d` command is:

```
{"status": "done", "dbgcmd": "delete"}
```

The debugger client can use these two commands to implement more complex functionality. For example, adding a condition to an existed unconditional breakpoint, or updating the expression of an existing conditional breakpoint.

Running the program

These are the debugging commands for running the program, including `continue`, `step over`, `step into`, and `step out`.

- `c`: Continue running the program

This resumes the running of the debugged TPS procedure. This continues to run until it reaches the next breakpoint.

The response to a `c` command is:

```
{"status": "running", "dbgcmd": "continue"}
```

- `n`: Step over (or “next”)

This resumes the running of the debugged TPS procedure. This stops when the beginning of the next source line is reached.

The response to the `n` command is:

```
{"status": "running", "dbgcmd": "step-over"}
```

- `s`: Step into

This resumes the running of the debugged TPS procedure. This stops when the beginning of the next source line is reached, if the next source line is not a procedure invocation. If it is, then it stops at the first command of the invoked procedure.

You cannot step into Tcl’s built-in commands or extension commands that are implemented in C/C++. The behavior is the same as “step over.”

You can step into an “unknown” procedure that is defined in `$HCIR00T/tcl/lib/tcl8.6/init.tcl`. This means the procedure to step into is not loaded yet and the auto-loading handler is triggered. You must locate the file in which the procedure is defined and re-run `hcitptest` with a `source` command added to the `-D` option’s Tcl script argument.

The response to the `s` command is:

```
{"status": "running", "dbgcmd": "step-into"}
```

- `o`: Step out

This resumes the running of the debugged TPS procedure until the current procedure exits.

The response to the `o` command is:

```
{"status": "running", "dbgcmd": "step-out"}
```

- `q`: Quit debugging session

This removes all breakpoints, then resumes the running of the debugged TPS procedure until the program ends.

The response to the `q` command is:

```
{"status": "running", "dbgcmd": "quit"}
```

After this command, when the program `hcitptest` ends, the connection to the Tcl debugger server is closed.

Data manipulation

Use the Tcl `set` command to change the value of a visible variable. For example:

```
set varName value
```

The response to the `set` command is:

```
{"status": "done", "dbgcmd": "set i 10", "result": "10"}
```

The UPoC Java package

JVM is part of the system package. You must have JDK to compile Java codes. JDK is not part of the system installation. To use the Java UPoC feature, you must download JDK 6.

Note: During the migration to CIS 6.2 and later versions, the Java UPoCs must be re-compiled.

The Java classes in the system's UPoC Java package provide facilities that you can use to manipulate a message within the system engine.

You can customize and extend the behavior of the system engine by specifying user code fragments. The engine runs these fragments at well-defined points in the processing of messages. Sometimes referred to as "hooks" or "user exits," these are formally known as UPoCs (User Points of Control).

As mentioned in the last section, the engine directly invokes UPoC code segments as Tcl procedures. To enable supports for UPoCs to written in Java, the engine uses Java Native Interface (JNI) to run Java codes.

Although there are many UPoCs, there are only three interface styles between the engine and the user code:

- TPS: Controls/extends message movement through the engine.
- TrxID: Determines a transaction code (routing identifier) for a given message.
- XLT: Transforms certain fields within a message as part of a transformation specification.

In the Java API, each of these styles is represented as a class which can be extended by the interface developer. To write useful code, the Java programmer makes use of several other classes that are provided within the package `com.cloverleaf.Cloverleaf.upoc`.

This table lists these classes and the objects that they represent:

Class	Objects represented
Message	A message passing through the engine. You can modify message content, message metadata, and create new messages.
DispositionList	A set of instructions telling the engine what to do with specified messages. The programmer can create new disposition lists.
GRM	A Generalized Record Manager, which is a way of addressing specific fields within a message by their logical names. GRM has subclasses for the different types of message layouts supported by the system: FRL, HRL, and VRL messages. The Java programmer can create an empty GRM, set and retrieve field contents from the GRM, construct a GRM from a message, and construct a message from a GRM.

Class	Objects represented
XPM	A Translation Pseudo Machine that provides GRM access to a pair of messages. These are the input and output messages of a translation which the engine is processing. An XPM also provides additional functionality not available through a GRM.
PropertyTree	<p>A hierarchical properties set. This is similar to a Hashtable or Properties object, but with certain restrictions and certain additional convenience methods. A PropertyTree is used to represent data structures that are not fixed at engine compile time.</p> <p>For example, the engine can be configured to construct an instance of a newly-developed subclass of TPS without the requirement for an engine recompilation. The new subclass might require certain configuration information. This can be specified in the Network Configuration definition, read by the engine, and passed as a PropertyTree constructor argument to the new subclass.</p> <p>Another example is in the facility to extend the metadata associated with each message to include additional metadata items not compiled into the engine (USERDATA). The USERDATA of a message is available to the Java programmer as a PropertyTree object.</p>

The methods that are used by the Java programmer also must refer to the system run-time environment within which they are running. A class `CloverEnv` represents the run-time environment.

Exceptions which stem from incorrect use of the API are represented by exceptions of class `CloverLeafException`.

Java equivalents

The available methods and the Java equivalents of the Tcl commands of `Message`, `DispositionList`, `PropertyTree`, and so on, are located in the javadocs. This is installed at `%HCIR00T%\docs\CloverleafJavaUpocAPI`.

There are also Java docs for the Java driver at `%HCIR00T%\docs\CloverleafJavaDriverAPI`.

Script UPoC Java Embedded Python

Java Embedded Python (JEP) support is in Cloverleaf UPoC. To use JEP, you must have the latest Python3.x installed on your machine.

The JEP package is located in `$HCIRoot/python/jep`.

Note: Python2.x UPoC is implemented with Jython.

On AIX machines, Cloverleaf is 32-bit. For this platform, you must install 32-bit Python3.x for JEP.

Note: On Windows, you must run the Python installer as "administrator".

JEP embeds CPython in Java through the Java Native Interface (JNI).

The benefits of embedding CPython in a JVM include:

- Using the built-in (native) Python interpreter may be much faster than alternatives.
- Python is mature, well supported, and well documented.
- Access to high quality Python modules, both built-in CPython extensions and Python-based.
- Compilers and assorted Python tools are as mature as the language.
- Python is an interpreted language, enabling scripting of established Java code without requiring recompilation.
- Both Java and Python are cross platform, enabling deployment to different operating systems.

The configuration for JEP UPoC is the same as JavaScript and Jython UPoC.

Java UPoC configurations use these formats.

Note: “`{{key> <value>...}}`” are optional arguments provided by the user.

- TPS:

```
{ PROCS cljTPS }
{ ARGS {{{CLASS class_name} {key value} ...} } }
```

- TrxID:

```
{ PROC {cljTrxid {{{CLASS ScriptTrxid}}} }
{ ARGS {{{key value} ...} } }
```

- Xlate:

```
cljXLTStrings {XLT_STYLE SINGLE} {CLASS class_name} {key
value} ...
```

LANG, FILE and FUNC, and SCRIPT keys

For JEP UPoC, the LANG, FILE, and FUNC keys, or SCRIPT are the same as other script languages. For example, javascript and jython.

Key	Description
LANG	Case-insensitive scripting language name. For Python JEP, the language is “jep”.
FILE	A bare file name without a directory. For example, <code>basename</code> . The file is searched for in the <code>scripts</code> subdirectories, under the current site, master site, and root directories. For JavaScript, the file names must have a <code>.js</code> extension. For Python, the extension is <code>.py</code> .
FUNC	Function name.
SCRIPT	Code snippet which is input directly as an argument.

Architecture with UPoC Java

There is one JVM loaded per engine process, and is loaded when the first Java UPOC command is invoked. This stays loaded in memory until the engine process terminates.

When a UPoC Java class is first used by the engine, an instance of the class is instantiated in the JVM; subsequent uses of the same class in the same thread make use of the same instance. These instances stay in the JVM until the thread that instantiated these classes die. This also means that different threads instantiate different instances, even if they make use of the same UPoC class.

The scope of static variables is the lifetime of the process, and the scope of instance variables is the lifetime of the thread.

The extendable classes (TPS, TrxID, and the XLT subclasses) share a common base class, `Upoc`, and a common approach to constructors.

Constructors and userArgs

`Upoc` contains the instance variable `protected PropertyTree userArgs;`.

This provides two constructors:

```
protected Upoc () throws CLOVERLEAFException
protected Upoc (CloverEnv cloverEnv, PropertyTree userArgs)
               throws CLOVERLEAFException
```

When instantiating a user-defined class, the engine checks to see if a constructor accepting a `CloverEnv` and `PropertyTree` argument has been defined.

- If it has, then the class is instantiated using that constructor.
- If it has not, then the constructor with no arguments is invoked and the `userArgs` member of the new object is set.

Instances and construction

UPoCs that are configured in different threads in the engine are always instantiated as separate instances.

UPoCs that are configured with different user arguments are always instantiated as separate instances.

UPoCs of the same class which are configured in the same thread with the same user arguments are instantiated as a single instance.

For example, a logging/audit type TPS is configured for both inbound and outbound on the same thread with the same user arguments. This is instantiated only once.

Destroy method

A `destroy` method can be defined to handle shutdown requirements. This includes a clean up of any resources which the UPoC may have allocated. For example, connections to external systems.

This method is called on shutdown of the system thread that created the UPoC. Depending on the behavior of the Garbage Collector in the JVM. This can be significantly earlier than any Java finish method is called.

Use of class (static) variables

The same UPoC class can be configured on multiple threads in the system, and the use of static variables to permit communication between threads. There should generally be no thread-safety issues with the use of static variables, because of the co-operative multi-threading algorithms of the system.

Deliberate use of static variables to communicate between threads requires care on behalf of the Java programmer. This can lead to problems which are difficult to debug or reproduce.

Example

This code demonstrates the properties in this section:

```
public class CountMessagesTPS extends TPS
{
    public static int global_cnt = 0;
```

```

public int local_cnt = 0;
public DispositionList process(CloverEnv cloverEnv, String context, String mode, Message msg)
throws CloverleafException
{
    DispositionList dl = new DispositionList();
    if (mode.equals("run")) {
        System.out.println("global_cnt: " + global_cnt++);
        System.out.println("local_cnt: " + local_cnt++);
    }
    return dl;
}
}

```

When this is set as the Inbound Data TPS UPoC of the Inbound thread and Outbound Data TPS UPoC of the outbound thread, this message is shown in the log as the engine processes messages:

```

Inbound Thread: global_cnt: 0
Inbound Thread: local_cnt: 0
Outbound Thread: global_cnt: 1
Outbound Thread: local_cnt: 0
Inbound Thread: global_cnt: 2
Inbound Thread: local_cnt: 1
Outbound Thread: global_cnt: 3
Outbound Thread: local_cnt: 1
Inbound Thread: global_cnt: 4
Inbound Thread: local_cnt: 2
Outbound Thread: global_cnt: 5
Outbound Thread: local_cnt: 2

```

Generic Java Driver V2

The initial version of the Generic Java Driver (GJD) is limited in that only one GJD can exist in a process. You cannot use a Java UPoC in the process as the UPoC. Driver setup fails when it tries to start a JVM.

The Generic Java Driver provides a framework for Java code that can be used to implement the functionality of a system driver.

Features of the Generic Java Driver include:

- The GJD can permit more than one GJD protocol thread in one process.
- The use of Java UPoCs in the process is supported.
- There is no restriction on the GJD and Java UPoC start order.
- GJD protocol threads permit shutdown and restart when possible.

The changes that are made in V2 are backwards compatible with the prior version of the GJD.

Environment variable "CLASSPATH"

Any existing class path specified in the environment variable CLASSPATH is added to the end of the user-specified entries by the `ini` files. This conforms with prior use of the variable by the Java UPoC code.

Non-engine Java UPoC

When an idle Java UPoC is called, in `hcitcl` the JVM that loads is the one that is not modified by the changes contained in GJD V2.

Using multiple javadriver threads in one process

In earlier versions, restarting a java thread would fail to load `lib` files, causing a resource recycling issue.

You can use multiple Java driver threads in one process by setting a `USE_LOADER=true` directive in the `pni` file.

This cannot be used on the GUI side.

When the GUI saves NetConfig:

- If one Java driver is found in one process, then the `use_loader` key is removed.
- If multiple Java drivers are found in one process, then the original value of `use_loader` is retained, if there is any. Otherwise, `use_loader=true` is updated in the `pni` file.

A warning message then opens, stating that multiple javadriver threads are detected in one process.

The engine with the directive can have each thread load a class in its own thread local `urlClassLoader` without interfering with each other on the process level. However, you can create in your own code a `ClassLoader` that can be shared among threads. This results in classes that will not be recycled after the CIS engine thread is down. In this case, you must carefully manage the `ClassLoader`. In a similar manner, this can also happen in the Spring context. This opens an additional warning message.

GJD V2 design

In Java, there can be only one JVM per process, so a single JVM is started by the driver code. Alternatively, the UPoC code must set the class path for both of them. Even then, you cannot destroy a JVM and create another without ending the process. Thus, the JVM becomes a global static in the engine.

When a JVM is started, the thread that started it becomes the main Java main thread. That thread cannot be terminated or the JVM errors. The starting of the JVM cannot be finished in an engine thread. It is only finished in its own sub-thread. That sub-thread exists until the engine process is shut down.

In the initial design, the JVM parameters were contained in an `.ini` file named for the protocol thread.

An issue results when more than one UPoC and GJD protocol thread are in use.

When the UPoC starts, the thread might not contain a GJD protocol driver. The JVM parameters in V2 are common to the process. A new control file is introduced that is named for the process instead of a thread. All GJD protocol threads still have their `thread_name.ini` file. The JVM section of those files might contain only an "include" directive that points to the `process_name.pni` file.

The driver section of `thread_name.ini` remains unchanged as it sets up the GJD instance in the JVM. When the UPoC code starts, first the `process_name.pni` file is directly accessed because there may not be any `thread_name.ini` for that engine thread. (Remember that Java UPoCs can be used in translations, so the thread name can be the `xlate` thread and not a protocol thread.)

During JVM activation, its parameters are obtained first from `process_name.pni`, if it exists. If it does not exist, then `thread_name.ini` is used, if it exists. If neither of these files exist, then it is assumed that the parameters are the defaults for UPoC use.

The `process_name.pni` file contains any user settings, such as configuring the JVM for Java debugging operations.

In earlier versions, the `java_uccs` directories were searched for any jar files and they were added to the default class path. Now, this searching feature has been added for any user-specified directory in the class path.

When there are more than one GJD in the process, the directive `USE_LOADER=true` must be added to the JVM section of `process_name.pni`. This entry causes a `URLClassLoader` instance to be used for loading the user's classes.

If it is not there, then the default JVM loader is used. Same-named classes that are used in different drivers are not unique. Protocol thread restarts can reuse previously loaded classes whose memory may not be garbage collected.

With multiple GJD threads in the process, indicated by `USE_LOADER=true`, the user class loading is modified so that each GJD uses its own instance of `URLClassLoader`. Because it becomes the root loader for all GJD's classes and the loader is part of a class, if the same class is used in more than one GJD they are not the same class.

When a GJD thread is shut down, the instance of the loader is destroyed. All classes that it loaded are cleaned up by the JVM garbage collector. If the GJD protocol thread is restarted, then a new instance of the loader is created. No class instances are reused from the previous running of the thread.

A Java driver shared mode restriction is that using its own classloader of the java app must be issued in shared mode of the Java driver.

There are problems in using the shared JVM in that the `URLClassLoader` must be used for all class loading. The shared JVM could also be the root loader for such in the GJD. Not all available code has this feature. For example, Tomcat, during start up, ignores this loader and uses the system loader. In this case, two instances of Tomcat cannot exist in a single process. The protocol thread cannot be restarted if it is shut down. In general, you cannot know this problem exists without using the code and trying a protocol thread restart and looking for errors. For user-written code that uses things that also cannot be run as applications, there should be no problems.

Multiple javadriver threads

You can use multiple Java driver threads in one process by setting a `USE_LOADER=true` directive in the `pni` file.

This cannot be used on the GUI side.

When the GUI saves NetConfig:

- If one Java driver is found in one process, then the `use_loader` key is removed.
- If multiple Java drivers are found in one process, then the original value of `use_loader` is retained, if there is any; otherwise, `use_loader=true` is updated in the `pni` file.

A warning message then opens, stating that multiple javadriver threads are detected in one process.

The engine with the directive can have each thread load a class in its own thread local `urlclassloader` without interfering with each other on the process level. However, you can create in your own code a `classloader` that can be shared among threads. This results in classes that will not be recycled after the CIS engine thread is down. In this case, you must carefully manage the `classloader`. In a similar manner, this can also happen in the Spring context. This opens an additional warning message..

thread_name.ini and process_name.ini

All existing `thread_name.ini` files work as long as no Java UPoCs or another GJD thread is added to the engine process.

`thread_name.ini` should contain only the "include" directive when a Java UPoC is used in a process that has at least one GJD in the JVM section of `process_name.pni`. This also applies when there is more than one GJD in the process. If this is not finished and the UPoC code starts before the GJD code, then the UPoC works but the GJD creates an error.

When the `thread_name.ini` contains the JVM parameters (a single GJD in the process), the class path values can now contain one or more `path/*.jar` entries.

The `process_name.pni` should be placed in the site directory. It requires only the JVM section as it is not used by GJD protocol threads. The entries that are defined in the prior version can be used. The class path should contain entries for all GJD protocol drivers that are used in the process. UPoC-specific items are not stated as long as all UPoC class and jar files are located in the `java_uccs` directories. They are added to the class path by default.

The start directory can be specified; only one can be specified. Its use is the same as in the prior version except that all Java drivers share that one directory. That is, it can be specified using the relative path "dot" format when defining class path entries.

Class path "path/*.jar"

In earlier versions, all `jar` files had to be explicitly listed using the class path directive in the `ini` file. To reduce the number of these and the possibility of leaving one out, you can use `path/*.jar`. When this is used, all `jar` files in the specified path directory are added to the JVM class path. If the start directory is specified, then the relative "dot" path can be used.

For example, if all `jar` files must be in the directory `my_jars` located in the start directory, then `./my_jars/*.jar` places all `jar` files on the class path. The order of the `jar` file names that replace this entry in the class path is not defined.

Generic Java driver V2 FAQ

This table lists frequently asked questions regarding the generic Java Driver:

FAQ	Best practice
Must I change anything for the Java UPoCs I currently use?	Not when you are not using a GJD in the same process. Even then, in most cases there are no UPoC-related V2 changes.
Can I still use environment variables to set my Java UPoC options?	No. You must use the <code>process_name.pni</code> file to set them even if there is no GJD in the process.

FAQ

Best practice

How do I debug JMX and my Java UPoCs?

Create a `process_name.pni` and set the options you require for the JVM in it.

Can I now shut down and restart threads with Java UPoCs without problems?

The best usage is to stop the process as before. The loaded Java classes are not removed from the JVM when a thread is shut down.

If I use a Java UPoC in a GJD protocol thread, then is there anything special I must do?

No. All of the previous defaults are set for the UPoC in the JVM that is started in the process.

Can I run my Java UPoC in `hcitcl`?

If you did in earlier versions, then you still can. When the UPoC is run this way, it uses environment variables to set its options as in the previous versions. No process name is available in this mode, so JVM setup cannot access the `ini` file.

I am currently using an earlier GJD; must I change anything for V2?

No. Previous versions of GJD protocol threads work without modification as long as you do not add a second GJD protocol thread to the same process.

What makes a loaded Java class unique?

Java uses the `package.class_name` and the instance of the loader that loaded the class to define a class. So, when a class with the same `package.class` name is loaded by two different loaders, they are not the same class. This is important when you use a shared JVM.

Can I use a single `package.class` in both a UPoC and a GJD?

If the JVM is not shared, then the class that is used by both is the same. In other words, the class should have no static members that are modified by both the UPoC and the GJD code.

Is the `process_name.pni` file required in all cases?

No. If a single GJD is used in the process and no UPoC requires the file, then `thread_name.ini` can contain the JVM parameters instead of `process_name.pni`.

What goes into the `process_name.pni`?

It should contain only the JVM section as described for the `thread_name.ini`. That is, the parameters that are required when a JVM starts. When there is more than one GJD running in the process, the parameters are the union of what is in their `thead_name.ini` files. Required files for all Java UPoCs being used are also included.

In what order are `ini` files processed?

When the JVM is start first the `process_name.pni` is used, if found there. If it is not found, then the `thread_name.ini` is attempted and used to set up the JVM, if found. If neither of these files are found, then only the UPoC JVM setup is finished. If one of these files is found, then the UPoC setup is added to the end of the parameters set by the file.

FAQ

What does "true" mean?

Can I have a class path that is unique to the GJD instance?

Can "shared" be used when there is only one GJD in the process?

What is new in the JVM section of the `ini` files?

Can I now stop and start a GJD protocol thread?

Best practice

When `USE_LOADER` is set to "true", each GJD class is loaded by its own class loader. The class path of each is usually the same. This makes two classes with the same `package.name` unique in the JVM. Because they are unique, any statics that are used in the class are also unique.

Yes, but be cautious. Usually, the `thread_name.ini` only contains an "include" that points to the `process_name.pni` file.

When the unique loader is used ("shared" is being used), the class path for that loader uses only the `thread_name.ini`. Thus, if you add more class path entries in the `thread_name.ini`, then they are used only for that loader instance. In this way, you can set up common things in the `process_name.pni` and add unique jar files for the GJD instance by doing this.

Yes. If the code permits it, then this lets that instance of the GJD to be shut down and restarted. This can be an advantage if you are still changing the Java code. You can shut down, change a jar, and restart using the new code. If the implementation of the GJD Java does not permit shut down/restart, then there is no requirement to set things to "shared".

The new parameter is `USE_LOADER`. This should be commented out, not be there, or set to "true."

This depends on what code is being loaded and how it is loaded. Some things, for example, Tomcat, uses the system loader no matter how they are started so cannot be shut down. Neither can things that conflict with classes loaded by Tomcat be used in the same JVM. Most things should permit it, but you must try it to find out if it works. If "shared" is not set, then the general answer is "no." Objects are not removed from the JVM when the thread is shut down. If a Java UPoC is being used, then the answer is always "no."

FAQ

Does the supplied general driver code support more than one GJD in the process?

Best practice

Only if you have set "shared", although perhaps not even then. If the supplied link class is loaded by the same loader, then it is not unique. It contains static variables that are used by the C code of the protocol driver. Because the class is not unique, these variables are identical and cannot hold the two values as required by two different GJD protocol drivers.

There are several ways around this:

- You can set "shared", which makes the link class unique for each driver instance.
- or
- You can re-factor the supplied source code so that it becomes unique and add that jar to the install. You can use the new names in the class parameters. This is the reason they are not fixed.
- or
- Use different processes for the GJD protocol threads.

Does setting shared always work?

No. Some Java code does not use the GJD loader after they begin running. For example, Tomcat switches to the system loader soon after it starts. Other large code collections may do the same. Even if the unique loader is used, other things may prevent protocol shut down and cause problems.

For example, another loader has a daemon thread that does not stop when you attempt to stop the server. Thus, there is no advantage to using "shared" for it. If there are strange errors or the thread does not shut down when "shared" is used, then move the GJD to its own process. There is no general way to predict when this is required.

What kind of errors can show up when using "shared"?

Currently, one known issue happens when an anonymous class is used for a new thread. This is the `$ class` that is made when the class compiles. For unknown reasons, you get "Illegal Access" for the `$ class` when creating the new thread object. To workaround, you can create a new independent class for the thread. You can also make the class with the anonymous class implement `Runnable` and move the run method into the class. It is yet to be determined whether other uses of anonymous classes also have this issue.

FAQ

If "shared" has problems, then why is it there?

Best practice

Using separate loaders (that is, "shared"), can be useful to reduce the number of processes being used in many cases. Not every GJD can use it until later. The main problem with using it is when odd errors happen. These show up early in the development process, so are not considered detrimental. The advantages outweigh the disadvantages.

Forgetting "shared", why was this version of the GJD created?

To permit the use of Java UPoCs in a process that contains a GJD protocol driver.

Directories and sample files

There are two directories for siteProto and each newly-created site:

- `$HCISITEDIR/java_uccs/src`
This is for keeping the Java UPoC *.java source code.
- `$HCISITEDIR/java_uccs`
This is for keeping the Java UPoC *.class files.

Additional directories

Additional directories include:

- `/integrator/java_uccs`
- `/integrator/java_uccs/src`
- `/integrator/java_uccs/src/templates` This directory also contains these files:
 - `Tps.java`
 - `Trxid.java`
 - `XltCall.java`
 - `XltObject.java`
 - `XltObjects.java`
 - `XltString.java`
 - `XltStrings.java`

Use this file as a template to create your Java UPoC code: `/cis6.2/integrator/java_uccs/src/samples`.

This directory also contains these files:

- `EdfTPS.java`
- `HL7TPS.java`

- TestXlateVectorPre.java
- Trxid.java
- X12TPS.java
- XlateSinglePre.java

These sample programs show how to use the Java UPoC feature.

Environment setup

As you set up the environment, you must set the CLASSPATH.

When `setroot` and `setsite` are run, `integrator/java_uccs` and `integrator/<site name>/` are prepended to CLASSPATH. If your classes are located in directories other than these, then you must set the class path so user-defined classes can be located by the engine.

You can update the CLASSPATH variable directly or use the environment variable `QDXI_CLASSPATH` to specify the location of user `.class` files. Values in `QDXI_CLASSPATH` are prepended to the usual class path when the JVM is loaded.

To pass extra init options when starting the JVM, you can set them in the process configuration from the GUI. To do this, select **Process > Configure** on the **Network Configurator**. This opens the **Process Configuration** dialog box. Select the target process, and then select the **Java Driver** tab.

The legacy `CLJAVA_INIT` environment variable can only set init options for the TPS testing tool. The value of the variable must be parsable as a Tcl list.

Java: Transaction ID determination interface

Java programmers can write users code fragments to determine a transaction code, that is, routing identifier, for a given message.

Apart from its constructors and `destroy` method, the `TrxID` class has one `process` method.

```
public String process(CloverEnv cloverEnv, Message msg)
    throws CLOVERLEAFException;
```

Examples

This example returns a transaction ID that is based on the predefined position, looking for a transaction ID that is `ADM` or `XFR`. If no such ID is found, then `NULL` is returned.

```
/*
 * $Id:$
```

```

* Infor
*/
import com.healthvision.CLOVERLEAF.upoc.*;
public class Trixid extends Trxid {
/**
* process a Trxid
*/
public String process(CloverEnv cloverEnv, Message msg)
    throws CLOVERLEAFException {
    String trxid = null;
    System.out.println("in TestTrxid");
    //get the message data to parse trixid
    String messageData = msg.getContent();
    String findTrxid = messageData.substring(5, 8);
    System.out.println("Trxid = " + findTrxid);
    //determine whether trxid had found
    if ( findTrxid.compareTo("ADM") == 0 || \
        findTrxid.compareTo("XFR") == 0) {
        trxid = new String("ADMANDXFR");
    } else trxid = findTrxid;
    System.out.println("End of TestTrxid");
    return trxid;
}
}

```

This example returns a transaction ID that is based on the message metadata Userdata. It looks for a value that has been stored under a TRXID key. If no such ID has been stored, then NULL is returned.

```

import com.healthvision.CLOVERLEAF.upoc.*;
public class SampleTrxid extends Trxid
{
    public String process(CloverEnv cloverEnv, Message msg)
    throws CLOVERLEAFException
    {
        String returnString = msg.getUserdata().getString("TRXID");
        if (returnString == null)
            returnString = "NULL";

        return returnString;
    }
}

```

Java: Code fragment interface

Java programmers can write user code fragments to transform certain fields within a message as part of a transformation specification. Translate classes can be used for pre, post, or invocation actions.

Translation actions and UPoC SubClasses

There are five UPoC subclasses concerned with user code in the context of message translation.

A common use for user code in translations is to manipulate the fields that are associated with an xlate operation before that operation is performed by the engine. For example, COPY, MATH, or TABLE. Four UPoC subclasses are provided for this purpose. Two subclasses deal with the fields as strings, and two which use different classes represent fields. depending on the type of field.

- Number of `Inval` fields: 1
Treat all fields as strings?
 - `xlateString`: Yes
 - `xlateObject`: No
- Number of `Inval` fields: More than 1
Treat all fields as strings?
 - `xlateStrings`: Yes
 - `xlateObjects`: No

The required interface style, which method is invoked by the engine, is determined by the options configured in the translation specification by the Translation Configurator.

A fifth UPoC subclass for translation, `XLTCall`, is provided for Translate CALL operations. In this case, the `CALL` method is invoked by the engine.

Each of the classes has one method. All are given an XPM object as an input parameter. All except `XLTCall` are given a representation of the input values to the Xlate action.

- `XLString` has an `xlateString` method, taking a single string input value and returning a single object output.
- `XLStrings` has an `xlateStrings` method, taking a vector of string input values and returning a Vector of Object output values.
- `XLObject` has an `xlateObject` method, taking a single object input value and returning an object output. Conversion between system field types and Java classes is performed automatically as specified in the API documentation of the Datum class.
- `XLObjects` has an `xlateObjects` method, taking a vector of object input values and returning a Vector of Object output values. Conversion between system field types and Java classes is performed automatically as specified in the API documentation of the Datum class.
- `XLTCall` has a `process` method that has no return and is given no input values. To do anything, the implementation of this method must use the methods of the XPM object that is passed in.

Examples

This example extracts the input field value and address, prints it to standard output and returns a hard-coded value.

```
/*
 * $Id:$
 * Infor
 */

import com.infor.CLOVERLEAF.upoc.*;
import java.util.*;
public class XlateSinglePre extends XLString {
/**
 * process a single value call
 */
public Object xlateString (CloverEnv cloverEnv, Xpm xpm, String inVal)
throws CLOVERLEAFException {

    /*
     * add something to the string
```

```

    */
    Vector v = new Vector();
    v.add(xpm.getInAddresses());

    System.out.println("In TestXlateSingPre proc");
    System.out.println("Input Address: " + v.toString() + \
        "Input Value: " + inVal);
    String outVal = new String( "Test Xlate PreProc with \
        single value parameter");
    System.out.println("Output Address: " + v.toString() + \
        "Output Value: " + outVal);
    System.out.println("End of TestXlateSingPre proc");
    return outVal;

}
}

```

This example operates on multiple values. The input and return values are vectors of strings.

```

/*
 * $Id:$
 *Infor
 */
import com.healthvision.CLOVERLEAF.upoc.*;
import java.util.*;

/*
 * this is a test for multi-valued call with strings
 * and fetch from xpm
 */

public class TestXlateVectorPre extends XLTStrings {

    /**
     * return an array of out values
     */
    public Vector xlateStrings (CloverEnv cloverEnv, Xpm xpm, Vector inVals)
    throws CLOVERLEAFException {

        Vector outputStrings = new Vector();
        System.out.println("In TestXlateVectorPre Proc");
        Enumeration e = xpm.getInAddresses().elements();
        System.out.println("Input Addresses:");
        System.out.println("-----");
        for (; e.hasMoreElements();) {
            Object o = e.nextElement();
            System.out.println("path="+o + " ");
            System.out.print("xpmfetch=");
            for (Enumeration e2 = xpm.getStrings(o.toString()).elements();
                e2.hasMoreElements();) {
                Object o2 = e2.nextElement();
                System.out.print("[ "+o2+" ] ");
                /*
                 * add to output list
                 */
                outputStrings.add(o2);
            }

            System.out.println();
        }
        System.out.println("In TestXlateVectorPre Proc");
        return outputStrings;
    }
}

```

This example ignores the input field value, and replaces it with a hard-coded string.

```
package sample;
import com.healthvision.CLOVERLEAF.upoc.*;

public class SampleXLT extends XLTString
{
    public String xlateString(CloverEnv env, \
                               Xpm xpm, \
                               String inVal)
    {
        throws CLOVERLEAFException
    }
    return "This is from Java sample.SampleXLT";
}
```

This example takes a property from its constructor argument. It then uses it to split a single input field into multiple output fields. Each field is no bigger than a maximum chunk size.

```
package sample;
import java.util.*;
import com.healthvision.CLOVERLEAF.upoc.*;
/**
 * <code>Split_Text_XLT</code> is used because in
 * messages many items consist of multiple lines of fixed
 * size whereas in the back office they are single strings
 */
public class Split_Text_XLT extends XLTStrings
{
    int chunkSize = 35;
    public Split_Text_XLT(PropertyTree userArgs)
        throws CLOVERLEAFException
    {
        super(userArgs);
        String chunkString = userArgs.getString("CHUNKSIZE");
        if(chunkString == null)
        {
            System.out.println("No CHUNKSIZE ?? using default: "
                               + chunkSize);
        }
        else
        {
            try {
                chunkSize = Integer.parseInt(chunkString);
            }
            catch(NumberFormatException e)
            {
                throw new CLOVERLEAFException\
                    ("CHUNKSIZE must be an integer");
            }
        }
    }

    public Vector xlateStrings(CloverEnv cloverEnv,
                               Xpm xpm,
                               Vector inVals)
        throws CLOVERLEAFException
    {
        if (inVals.size() == 0)
            // nothing to do
            return inVals;

        Vector outVals = new Vector();
        String inText = (String)inVals.firstElement();
        int length = inText.length();
        int startpos = 0;
        int endpos = chunkSize;
        while (endpos < length)
        {
            // add any full chunks to the outVals
        }
    }
}
```



```

        outVals.add(inText.substring(startpos,endpos));
        startpos += chunkSize;
        endpos += chunkSize;
    }
    if (startpos < length)
        outVals.add(inText.substring(startpos,length));
    if (outVals.size() == 0)
        outVals.add("");

    return outVals;
}

```

Java: TPS module interface

Java programmers can implement TPS-style programs by extending the TPS class. The engine invokes the user's `process` method.

- This method is passed a String mode that is one of Start, Run, or Time.
- This method is passed a message reference in the Run mode.
- This method returns a disposition list.
- A TPS can be used to implement additional protocol drivers using the UPoC protocol.

Apart from its constructors and `destroy` method, the TPS class has one method:

```

public DispositionList process (
    CloverEnv cloverEnv,
    String context,
    String mode,
    Message msg)
    throws CLOVERLEAFException

```

- `context` identifies which of the various TPS-style points of control is involved, for example, `sms_ib_data`.
- `mode` identifies whether the TPS is run in Start, Run, or Time mode. Requirements for Shutdown logic are handled by the `destroy` method inherited from the UPoC class.
- The message reference is a message that is passing through the engine. For Start and Time mode, the message can be null.
- The returned `DispositionList` instructs the engine what to do with the message that is passed into the invocation, and the message that is created during the invocation.

The context of a TPS instance, in most cases, is the same on every invocation of the `process` method. The exception being the case of a startup TPS which can also have context for `startup_send_ok` and `start up_send_fail`.

Examples

This example prints the content of the message that is passed in to standard output. It then creates a new message, and returns a disposition list. Then, it instructs the engine to process the new message and stop the one which was passed in.

```
package sample;
import com.healthvision.CLOVERLEAF.upoc.*;

public class MyTPS extends TPS
{
    public DispositionList process (
        CloverEnv cloverEnv,
        String context,
        String mode,
        Message msg)
        throws CLOVERLEAFException
    {
        DispositionList dispList = new DispositionList();
        if (msg == null)
            return dispList;

        System.out.println(msg.getContent());

        dispList.add(DispositionList.CONTINUE,
            cloverEnv.makeMessage("New message from MyTPS"));
        dispList.add(DispositionList.KILL, msg);
        return dispList;
    }
}
```

This example calls a hypothetical external class method that incorporates a business logic rule, passing the message content string. It stores the returned value in `userData metadata` . Then, it returns a disposition list instructing the engine to continue processing the message.

```
package sample;
import com.healthvision.CLOVERLEAF.upoc.*;
// Business logic is defined in a separate package
import hypothetical.businesslogic.stuff.BusinessRule;

public class SetTrxidTPS extends TPS
{
    public DispositionList process (
        CloverEnv cloverEnv,
        String context,
        String mode,
        Message msg)
        throws CLOVERLEAFException
    {
        DispositionList dispList = new DispositionList();
        if (msg == null)
            return dispList;

        PropertyTree userData = msg.getUserdata();

        // obtain a transaction routing code from the Business
        // Logic package using a hypothetical static method
        userData.put("TRXID", BusinessRule.getTrxid(msg.getContent()));
        msg.setUserdata(userData);

        dispList.add(DispositionList.CONTINUE, msg);
        return dispList;
    }
}
```

This example uses a Fixed Record Layout Manager, `frlm` (a subclass of `Grm`) to create a new message based on the one passed in. Then, it changes and adds some fields. This instructs the engine to process both the new message and the original.

```
package sample;
import java.util.*;
import com.healthvision.CLOVERLEAF.upoc.*;

public class GrmSampleTPS extends TPS
{
    public DispositionList process(CloverEnv cloverEnv, \
        String context, String mode, Message msg)
        throws CLOVERLEAFException
    {
        DispositionList dispList = new DispositionList();
        if (msg == null)
            return dispList;

        System.out.println(msg.getContent());
        Frlm frlm = msg.makeFrlm("SomeFrl.frl");

        // See what's in field1 now
        Vector f1Vector = frlm.getStrings("Field1");
        Enumeration e = f1Vector.elements();
        while (e.hasMoreElements())
            System.out.println(e.nextElement());

        // Now change field called Field2 using a String value
        frlm.setString("Field2", "hello from Java");

        // Now change a field called Field3, using
        // a String to set its first subfield
        // and an Integer to set its second subfield
        Vector v = new Vector();
        v.addElement("a value for sf1");
        v.addElement(new Integer(123));
        frlm.setObjects("Field3", v);

        dispList.add(DispositionList.CONTINUE,
            frlm.makeMessage());
        dispList.add(DispositionList.CONTINUE, msg);
        frlm.destroy();
        return dispList;
    }
}
```

This example uses a `Grm` to create a new Edifact message that is based on the message passed in. It prints field contents of the message that is passed in to standard output. It also modifies some field contents. Then, it returns a disposition list instructing the engine to process the new message and stop the original message.

```
/*
 * $Id:$
 * Infor
 */
import com.healthvision.CLOVERLEAF.upoc.*;
public class EdfTPS extends TPS

    /**
     * standard TPS constructors
     */
    public EdfTPS(CloverEnv cloverEnv, PropertyTree xArgs) {
        super(cloverEnv, xArgs);
    }

    public EdfTPS() { }
    /** process a TPS*/
    public DispositionList process(CloverEnv cloverEnv, \
        String context, String mode, Message msg)
```

```

throws CLOVERLEAFException {

DispositionList dl = new DispositionList();
/** If there is no message then just return*/
if (msg == null) return dl;

/*Create an Edifact message */
EdifactM grm = msg.makeEdifactM("97B",null,"MEDRPT");

/*To extract message field content with no subfield*/
String x = grm.getString("2(0).2(0).0(0).FCA(0).1#4471(0).[0].[0]");

/* To extract message field content with subfield */
Vector v = grm.getStrings("2(0).3(0).0(0).PNA(0).6#C816" );

/* Print it out to the log file */
System.out.println(x.toString());
for (Enumeration e = v.elements(); e.hasMoreElements();) {
System.out.print "["+e.nextElement().toString()+"] ";
}

Vector setValue = new Vector();
setValue.addElement(new String("ZZZZ"));
setValue.addElement(new String("XXXXXX,XXXXXXXX"));

/*Call method setString and setStrings to modify message fields constant*/
grm.setString("2(0).2(0).0(0).FCA(0).1#4471(0).[0].[0]", "TTTTTT");
grm.setStrings("2(0).3(0).0(0).PNA(0).6#C816",setValue);
grm.setString("2(0).3(0).0(0).DTM(0).1#C507(0).[1]", "YYYYYYYYYY");
grm.setString("2(0).3(0).3(0).0(0).RFF(0).1#C506(0).[1].[0]", \
"UUUUUUUUUU");

//test Grm.makMessage method
Message outMsg = grm.makeMessage();

//retire the original message
dl.add( DispositionList.KILL, msg );

//Continue with a new message
dl.add( DispositionList.CONTINUE, outMsg);

return dl;
}
}

```

This example uses a `Grm` to create a new X12 message that is based on the one passed in. It prints field contents of the message that is passed in to standard output. It also modifies some field contents. Then, it returns a disposition list instructing the engine to process the new message and stop the original message.

```

/*
 * $Id:$
 * Infor
 */
import com.healthvision.CLOVERLEAF.upoc.*;
public class X12TPS extends TPS
{
/**
 * standard TPS constructors
 */

public X12TPS(CloverEnv cloverEnv, PropertyTree xArgs) {
super(cloverEnv, xArgs);
}

public X12TPS() { }
/** process a TPS*/
public DispositionList process(CloverEnv cloverEnv, String context, \
String mode, Message msg)
throws CLOVERLEAFException {

```

```

DispositionList dl = new DispositionList();
/** If there is no message then just return*/
if (msg == null) return dl;
/*Create an X12 message */
X12M grm = msg.makeX12M("004010",null,"270" );

/* To extract message field content with no subfield*/
String x = grm.getString("1(0).1(0).0(0).NM1(0).3#1035");

/* To extract message field content with subfield */
Vector v = grm.getStrings("1(0).1(0).0(0).PRV(0).3#127(0)" );

/* Print it out to the log file */
System.out.println(x.toString());
for (Enumeration e = v.elements(); e.hasMoreElements();) {
System.out.print "["+e.nextElement().toString()+"] ";
}

/* call method setString and setStrings to modify the field content */

Vector setValue = new Vector(1);
setValue.addElement(new String("Infor X12 \
Upoc Testing"));grm.setStrings("1(0).1(0).0(0).NM1(0).3#1035", \
setValue);grm.setString("1(0).1(0).0(0).PRV(0).3#127(0)","0000000000");

/*Create another message*/
Message outMsg = grm.makeMessage();

//retire the original message
dl.add( DispositionList.KILL, msg );

//Continue with a new message
dl.add( DispositionList.CONTINUE, outMsg);
return dl;
}
}

```

This example uses a Grm to create a new HL7 message that is based on the one passed in. It prints field contents of the message that is passed in to standard output. It also modifies some field contents. Thenk it returns a disposition list instructing the engine to process the new message and stop the original message.

```

/*
 * $Id:$
 * Infor
 */
import com.healthvision.CLOVERLEAF.upoc.*;
public class Hl7TPS extends TPS
/**
 * standard TPS constructors
 */

    public Hl7TPS(CloverEnv cloverEnv, PropertyTree xArgs) \
        {super(cloverEnv, xArgs);
    }

    public Hl7TPS() { }
    /** process a TPS*/
    public DispositionList process(CloverEnv cloverEnv, \
        String context, String mode, Message msg)
        throws CLOVERLEAFException {

        DispositionList dl = new DispositionList();
        /** If there is no message then just return*/
        if (msg == null) return dl;

        /*Create an Hl7 message */
        Hl7M grm = msg.makeX12M("004010",null,"270" );

        /* To extract message field content with no subfield*/

```

```

String x = grm.getString("1(0).1(0).0(0).OBR.00238","00000");

/* To extract message field content with subfield */
Vector v = grm.getStrings("1(0).1(4).1(1).OBX(0).00573");

/* Print it out to the log file */
System.out.println(x.toString());
for (Enumeration e = v.elements(); e.hasMoreElements();) {
    System.out.print "["+e.nextElement().toString()+"] ";
}

/* call method setString and setStrings to modify the field content */

Vector setValue = new Vector(3);
setValue.addElement(new String("3333333333"));
setValue.addElement(new String("4"));
setValue.addElement(new String("Infor"));

grm.setString("1(0).1(0).0(0).OBR.00238","00000");
grm.setStrings("1(0).0(0).0(0).PID.00106(0)",setValue);

/*Create another message*/
Message outMsg = grm.makeMessage();

//retire the original message
dl.add( DispositionList.KILL, msg );

//Continue with a new message
dl.add( DispositionList.CONTINUE, outMsg);

return dl;
}
}

```

Helper classes

The system Java API also contains a series of Helper classes that provide the means of accessing and manipulating objects in the system environment.

The available methods and Java equivalents of the Tcl commands of Message, DispositionList, PropertyTree, and so on, are in the javadocs at %HCIR00T%\docs\CloverleafJavaUpocAPI. There are also Java cdocs for Java driver at %HCIR00T%\docs\CloverleafJavaDriverAPI.

Message class

The Message class provides methods to:

- Cause the creation and destruction of messages.
- Set and retrieve message content.
- Set and retrieve metadata properties of messages.

A Message object represents a reference to a message being passed through the system engine. The message itself belongs to the system engine.

There is no public constructor for a message; instead, a Message object is obtained from the system engine, represented as a `CloverEnv`.

The system engine communicates with a TPS writer primarily by Message objects.

- If a Java UPoC creates a new message, then it does not own the resources associated with that message.
- If a Java class creates a message in a TPS, then the engine must be instructed on how to treat that message. It does this with a `DispositionList` or by destroying it.

Failure to do one represents a resource leak.

The GRM class (Generic Record Manager) also provides a standard method to create a new message based on a GRM instance. Conversely, the Message class includes standard methods to create instances of GRM subclasses based on a message.

A Java UPoC can remember a reference to a message between method invocations, for example, using a class variable. The programmer must be aware that the Message object in Java is not the same thing as the underlying message in the engine. The message in the engine does not depend on Java reference counting of the Message object in Java.

For example, a Java TPS creates a message, and does not specify a disposition for that message when returning a disposition list. Then, the engine takes no action on that message. It is valid for the TPS to include the message in a later disposition list. This is a common technique often used to consolidate or match messages.

An exception results if a message that is added to one disposition list is added to the disposition list returned by a later invocation to process.

The underlying messages are processed by the engine, and possibly by user code that is written in languages other than Java. The life cycle of a message is not controlled by the JVM.

The message contents are accessed or set as a String or as a byte array. Message content is a valid UTF-8 format if the String methods are used. Conversion between the UTF-8 representation in the message and the String representation seen by the Java API programmer is performed by the system, not the JVM.

Note: ASCII characters in the range 1 to 127 are valid UTF-8 representations of themselves.

DispositionList

A disposition list is a means for a TPS to communicate instructions to the engine for message processing.

A disposition list has an `add` method. For example:

```
public void add(int disposition, Message msg)
               throws CLOVERLEAFException
```

The valid disposition codes are provided as static final members of the class with values:

- CONTINUE
- ERROR
- KILL
- KILLREPLY
- OVER

- PROTO
- SEND

An `Rm` is a record manager. It provides a means to make use of the parsing facilities of the system, and set or retrieve fields within a message using logical names. `Rm` methods are used from TPS's and `TrxID` classes using `GRM` and its subclasses, and from `XLT` (Translation) classes by `XPM`.

`Rm` provides two alternative styles for accessing field contents:

- In one style, all retrieved values are strings, and all stored objects are converted to `String` objects using their `toString()` method.
- In another style, retrieved values are objects of classes which depend on the system field type (for example, as specified in the `FRL` Configurator).

For example, an `ai` field is represented as a `BigInteger` object. The conversion is performed by methods of the datum class. Objects that are stored back into fields must be of classes which are supported by the datum conversion methods.

Type conversions that are used when converting from system types to Java classes are:

- From CIS type: `ai`
To Java class: `BigInteger`
- From CIS type: `br`
To Java class: not supported
- From CIS type: `ch`
To Java class: `String`
- From CIS type: `dt`
To Java class: `Date`
- From CIS type: `ed`
To Java class: `Date`
- From CIS type: `fd`
To Java class: `Date`
- From CIS type: `fe`
To Java class: `Date`
- From CIS type: `jd`
To Java class: `Date`
- From CIS type: `n1 - n10`
To Java class: not supported
- From CIS type: `nm`
To Java class: `BigDecimal`
- From CIS type: `null`
To Java class: `null`
- From CIS type: `st`
To Java class: `String`

- From CIS type: tm
To Java class: Date
- From CIS type: ts
To Java class: Date
- From CIS type: ut
To Java class: Date
- From CIS type: yd
To Java class: Date
- From CIS type: zd
To Java class: not supported
- From CIS type: usy
To Java class: String
- From CIS type: uan
To Java class: String
- From CIS type: cd
To Java class: BigDecimal
- From CIS type: swx
To Java class: String
- From CIS type: swy
To Java class: String
- From CIS type: swz
To Java class: String
- From CIS type: hex
To Java class: not supported

Type conversions that are used when converting from Java classes to system types are:

- From Java class: Integer
To CIS type: ai
- From Java class: BigInteger
To CIS type: ai
- From Java class: String
To CIS type: ch
- From Java class: Double
To CIS type: nm
- From Java class: Float
To CIS type: nm
- From Java class: BigDecimal
To CIS type: nm

- From Java class: `Date`
To CIS type: `ts`
- From Java class: `null`
To CIS type: `null`

GRM and its subclasses

A GRM is a Generalized Record Manager which can be created, accessed and destroyed by the Java API programmer. It provides a means to make use of the parsing facilities of the system, and set or retrieve fields within a message using logical names.

The supported subclasses of GRM are:

- `FrLM`
A Fixed Record Layout Manager that permits fields within a record to be set or retrieved by name. Similar to other GRM objects, an `FrLM` can be constructed from a `Message` and a new `Message` can be created from an `FrLM`.
- `VrLM`
A Variable Length Record Layout Manager that permits fields within a record to be set or retrieved by name. Similar to other GRM objects, a `VrLM` can be constructed from a `Message` and a new `Message` can be created from a `VrLM`.
- `HrLM`
An HRL Record Layout Manager that permits fields within a record to be set or retrieved by name. Similar to other GRM objects, an `HrLM` can be constructed from a `Message` and a new `Message` can be created from an `HrLM`.
- `Hl7M`
An HL7 Record Layout Manager that permits fields within a record to be set or retrieved by name. Similar to other GRM objects, an `Hl7M` can be constructed from a `Message` and a new `Message` can be created from an `Hl7M`.
- `XmLM`
An XML Document Layout Manager that permits tags within a document to be set or retrieved by name. Similar to other GRM objects, an `XmLM` can be constructed from a `Message` and a new `Message` can be created from an `XmLM`.
- `EdifactM`
An Edifact Layout Manager that permits fields within a message to be set or retrieved by name. Similar to other GRM objects, an `EdifactM` can be constructed from a `Message` and a new `Message` can be created from an `EdifactM`.
- `X12M`
An X12 Layout Manager that permits fields within a transaction set to be set or retrieved by name. Similar to other GRM objects, an `X12M` can be constructed from a `Message` and a new `Message` can be created from an `X12M`.

As with a message, a GRM represents a reference to resources which are managed by the engine. A GRM which is created should always be destroyed manually.

As with message instances, GRM references can be remembered and shared among code, for example, using static variables. The programmer must be aware that the life cycle of the underlying generalized record manager is not connected with Java reference counting.

For example, a TPS has called `Destroy` on a GRM. Subsequent references from another TPS which had a reference to the Java GRM object would raise exceptions.

The `CloverEnv` class provides standard methods which permit the API programmer to request new, empty GRM instances to be created by the system engine.

The `Message` class includes standard methods to create instances of GRM subclasses based on a message. Conversely the GRM class also provides a standard method to create a new message based on a GRM instance.

XPM

An XPM is a reference to the translation pseudo machine of the system. It also reflects the state of the pseudo machine at the point at which the associated `CALL`, `COPY`, or other translation action is performed.

The XPM class provides these types of functionality:

- An `Rm` representing both the input and output messages of the translation process. It also provides temporary variables that are referred to directly by the **Translation Configuration** dialog box.
- A set of message metadata representing the metadata of the output message. This is preinitialized by the engine to be the metadata of the input message.
- Access to the addresses of the fields, if any, which were specified in the source column of the action that is currently being processed.

Unlike GRM subclasses, an XPM is only created implicitly by the engine, not explicitly by Java code. As a corollary, an XPM cannot be destroyed by Java code.

XPM methods can only be used within the methods of an XLT subclass. This is when the engine provides an XPM as an argument to the `process` methods.

PropertyTree

An instance of the `PropertyTree` class represents a tree structure, in which each branch and leaf has a string key.

- The value that is associated with a branch key is itself an instance of `PropertyTree`.
- The value that is associated with a leaf key is a `String`.

Nesting is supported and sub-branches are addressed by concatenating keys with a dot ".".

In many ways, `PropertyTree` provides a similar interface to the familiar `Hashtable` class.

Differences in methods between `Hashtable` and `PropertyTree` are:

- Key values are type `String` not `Object`.
- The presence of "." in a key implies nesting. So `put("A.B" "something")` implies a branch `PropertyTree` with key `A` containing a leaf with key `B`. The branch `A` is automatically created if it does not exist.
- The values which can be put are limited to classes `PropertyTree` and `String`.
- A `get(String key)` method is provided. You can use `getBranch(String key)` or `getString(String key)`, depending on whether the key was expected to address a branch or a leaf.
- There is no `elements()` method.

MsgDataInputStream and MsgDataReader

An instance of `MsgDataInputStream` or `MsgDataReader` is used for reading message contents from external sources as streams.

- `MsgDataInputStream` reads data as byte streams.
- `MsgDataReader` reads data as character streams.

Abstractly, message content is a sequence of bytes or characters. When reading from external sources, they are bounded by one of these formats: RAW, NL (newline), EOF (end of file), LEN10 (length encoded).

There are two types of external sources: a `Message` object, or another `InputStream`. These classes provide the utility function, `readData()`, to extract message content from the underlying source using the specified formats.

MsgDataOutputStream and MsgDataWriter

An instance of `MsgDataOutputStream` or `MsgDataWriter` is used for writing message contents to external sources as streams.

- `MsgDataOutputStream` writes data as byte streams.
- `MsgDataWriter` writes data as character streams.

Abstractly, message content is a sequence of characters. When written out to external sources, they must be wrapped in a certain format (RAW, NL, EOF, LEN10).

There are two types of external sources: a `Message` object, or another `OutputStream`. These classes provide the utility function, `writeData()`, to wrap message content with the necessary boundary characters and write them into the underlying source.

Exceptions

Incorrect use of the system Helper classes by the API programmer results in exceptions. These are of the type `CloverleafException`, which extends `java.lang.Exception`.

The extendable methods of TPS, TrxID, and XLT are declared as throwing `CloverleafException`; therefore it is acceptable to let any errors be caught by the system itself.

These are treated in this manner:

- If the exception is generated during processing associated with a message, then a stack trace of the error is printed in the engine log. It is also stored in the message metadata as its error context. The message is then sent to the error database.
- If the exception is generated during processing that is not associated with a message, then a stack trace is printed in the log. No other action is taken. Typically, this is in Start or Time mode in a TPS.

A requirement to notify someone of exceptions relating to messages is often addressed by one of these:

- Configuring a system alert based on the error-count criterion of a thread or group of threads.
- Using a configurable exception file that is referred to from UPoC code in validation logic or catch blocks, using the `notifyException`.

`CloverleafException` has two specialized subclasses:

- `PropertyTree.PropertyTreeException` which deals with incorrect use of `PropertyTree` tree structures.
- `DispositionList.InvalidDispositionException` which deals with invalid dispositions.

Exception notification mechanism

The Java UPoC can invoke the notification mechanism of the system language-independent exception that is based on the `exceptions.cfg` file. This file holds a numbered list of exceptions and associates, each of them with the shutdown, error, and notification properties.

Through this mechanism, you can:

- Log events in a central exceptions log and in the engine output log
- Stop a system thread
- Stop a system process
- Notify a user by email or other means

Configuration

The `exceptions.cfg` configuration file consists of definitions of numbered exceptions. It can be stored at the root level or the site level.

This is the format and specification of `exceptions.cfg`:

```
[exception number]
stop=[0 - do not shutdown
      1 - shutdown at thread level
      2 - shutdown at process level
      3 - shutdown at site level]
restartAfter=[number of seconds]
description=[description]
notificationMethod=[TCL code to run to send notifications]
userInfo=[user information]
```

A specific notification mechanism can be configured for particular exceptions by providing a TCL expression to be evaluated. The most commonly used notification mechanism in practice is email.

In Windows, exceptions that are treated by this mechanism are logged centrally in `$HCIRoot\exception.log`. On other platforms, they are logged both centrally in `$HCIRoot\exception.log` and in the engine output log.

Usage

The Java API programmer can invoke this exception notification mechanism using the `notifyWarning` and `throwException` methods of `CloverEnv`:

```
public void notifyWarning (int exceptionNumber, String text)
public void throwException (int exceptionNumber, String text)
```

In this example:

- `throwException` throws `CloverleafException`.
- `notifyWarning` does not throw `CloverleafException`.

There are two versions for each of the above methods. One is intended for use in catch blocks. This takes an additional argument of the original (caught) exception, for inclusion in the reported stack trace.

Utility and convenience methods

`CloverEnv` provides utility methods which can assist the API programmer.

Log method

This method permits the API programmer to incorporate diagnostic or audit output in the output log of the system engine.

The log's verbosity level is controlled at run-time without the Java API programmer having to change code. The `log` method has a verbosity argument that determines whether it is actioned at run-time, depending on environment and engine settings.

Note: The method invocation is made even if no output is produced. This is subject to any dynamic compile optimization in-lining provided by the JVM. There is still some overhead that is associated with each log statement left in production code.

For example:

```
cloverEnv.log(50, "Message contents now >" +
    msg.getContent() + "<");
```

This produces a line in the engine log similar to this, if the verbosity environment is set for 50 or above:

```
23/04/09 16:52:05 - 50 - Message contents now >This is a message<
```

Tcl evaluation from Java

Tcl (Tool Command Language) is a richly-featured scripting language. This is the primary language used to write system UPoCs. Java API programmers can invoke a Tcl piece to take advantage of inbuilt Tcl functionality. Or, an existing Tcl procedure written for their site can be used.

Tcl works by evaluating commands that have zero or more arguments and optionally returning a value.

The `tclEval` method of `CloverEnv` requests the engine to evaluate a Tcl command and returns any return value. Any error that is generated in the Tcl evaluation causes a `CloverleafException` to be thrown.

Using Java code from Tcl UPoCs

A mixture of Java and Tcl UPoCs can be configured within the same interface. In some instances, a Tcl UPoC calls out to Java code instead of returning to the engine. At this point, it waits for the engine to invoke a Java UPoC later on the TPS stack.

The Java API does not include any special Java interface to be invoked in this way. Tcl extensions are available for Tcl script writers to call into Java TPS, TrxID, or XLTs, if a translation is in progress.

For the available methods and the Java equivalents of `Message`, `DispositionList`, `PropertyTree`, and other Tcl commands, these are in the javadocs installed in `%HCIR00T%\docs\CloverleafJavaUpocAPI`.

There are also Java docs for Java driver at `%HCIR00T%\docs\CloverleafJavaDriverAPI`.

Calling a Java TPS from Tcl

The relevant Tcl extensions are `makeUpocInstance`, `processTPS`, and `destroyUpocInstance`.

These can be used directly, or a pre-provided Tcl procedure `cLjTPS`, which is itself a TPS-style interface, can be used to wrap them. Usually, the wrapping is adequate. There are situations, though, where Java UPoC instances are created implicitly through the wrapper and are not destroyed until the Tcl interpreter shuts down.

`cLjTPS` requires that the ARGs include a key `CLASS` which has the fully-qualified name of the TPS subclass to be used. The ARGs keyed list must match any `PropertyTree` keys that the Java UPoC is expecting.

`cLjTPS` can be called directly. The curly bracket syntax in the keyed list is difficult at times, so a further wrapper is provided to wrap it.

```
proc jtpsRun {context class msgid {userArgs {}}} {  
    keylset userArgs CLASS $class  
    return [cLjTPS "CONTEXT $context" "MODE run" \  
        "MSGID $msgid" "ARGS {$userArgs}"]  
}
```

```

}
proc jtpsStart {context class {userArgs {}} } {
    keylset userArgs CLASS $class
    return [cljTPS "CONTEXT $context" "MODE start" \
        "MSGID $msgid" "ARGS {$userArgs}"]
}
proc jtpsTime {context class {userArgs {}} } {
    keylset userArgs CLASS $class
    return [cljTPS "CONTEXT $context" "MODE time" \
        "MSGID $msgid" "ARGS {$userArgs}"]
}
proc jtpsStop {context class {userArgs {}} } {
    keylset userArgs CLASS $class
    return [cljTPS "CONTEXT $context" "MODE stop" \
        "MSGID $msgid" "ARGS {$userArgs}"]
}

```

These can be used with no user arguments:

```
jtpsRun sms_ib_data sample.SampleTPS message0
```

With the user argument CHUNKSIZE:

```
keylset sample2Args CHUNKSIZE 30
jtpsRun sms_ib_data sample.Sample2TPS message0 $sample2Args
```

The return value is a disposition list. Depending on what is required, the caller could iterate through this and look at the individual messages. At this point, it is already in the correct format to be returned to the engine.

A new instance is created implicitly whenever different user arguments are specified for the first time. There is no requirement to invoke Start mode first if the Java TPS does not require to be called in Start mode for any reason.

Calling a Java TrxID from Tcl

The relevant Tcl extensions are `makeUpocInstance`, `processTrxID`, and `destroyUpocInstance`.

These can be used directly, or a pre-provided procedure `cljTrxid`, which is a TrxID style interface, can be used to wrap them. Usually, the wrapping is adequate. At times, though, Java UPoC instances that are created implicitly through the wrapper are not destroyed until the Tcl interpreter shuts down.

`cljTrxid` requires a first argument that is a keyed list. This list must include a key `CLASS` which has the fully-qualified name of the TrxID subclass to use. The keyed list must also match any PropertyTree keys that the Java UPoC is expecting.

`cljTrxid` can be called directly, or a further wrapper might be helpful to wrap it.

```

proc jtrxidProcess {class msgid {userArgs {}} } {
    keylset userArgs CLASS $class
    return [cljTrxid $userArgs $msgid]
}

```

This can be used with no user arguments.

```
jrxidProcess sample.SampleTrxid message0
```


The return value is a string.

Calling a Java XLT from Tcl

The relevant Tcl extensions are `makeUpocInstance`, `processXLT`, and `destroyUpocInstance`.

These can be used directly, or pre-provided procs `cljXLTObjects` and `cljXLTStrings`, which are themselves XLT style interfaces, can be used to wrap them. Usually, the wrapping is adequate. At times, though, the Java UPoC instances that are created implicitly through the wrapper are destroyed until the Tcl interpreter shuts down.

The procs require a single argument that is a keyed list. This must include a key `CLASS` which has the fully-qualified name of the XLT subclass to be used. They also require a key `XLT_STYLE` that determines which method of the Java XLT is called. The keyed list must also match any `PropertyTree` keys that the Java UPoC is expecting.

The relationship between the `XLT_STYLE` and the method is:

- Link procedure name: `cljXLTStrings`
 - `XLT_STYLE: CALL`
Method: `process`
 - `XLT_STYLE: SINGLE`
Method: `xlateString`
 - `XLT_STYLE: VECTOR`
Method: `xlateStrings`
- Link procedure name: `cljXLTObjects`
 - `XLT_STYLE: CALL`
Method: `process`
 - `XLT_STYLE: SINGLE`
Method: `xlateObject`
 - `XLT_STYLE: VECTOR`
Method: `xlateObjects`

A `SINGLE` style only results in a method invocation if there is at least one input value to the translate action.

A `VECTOR` style always result in a method invocation.

If there are no input values to the translate action, then the vector that is passed in has zero elements.

An example (ignore wrap-around and new lines) which calls the `xlateStrings` method of `lnysample.Split_Text_XLT` is:

```
cljXLTStrings {CLASS lnysample.Split_Text_XLT}
               {CHUNKSIZE 35}
               {XLT_STYLE VECTOR}
```

The XLT interface uses `upvars`. Therefore, if `cljXLT` procs are called from within another XLT, they directly set the `xlateOutVals`, and so on, in the calling procedure.

JDDK

The Generic Java Driver provides a framework where Java code can be used to implement the functionality of a system driver.

The Generic Java Driver differs from the existing UPoC type driver. It is fully threaded, provides several features not possible in a UPoC driver. This places no restrictions on the use of Java threads in the implementation.

Users of the Generic Java Driver can write Java applications against an API that:

- Encapsulates the driver.
- Deploys the applications.
- Configures the system thread to make use of them.

The Java Protocol uses the Java Driver API to interact with the system engine by a set of message exchange patterns.

The Java Driver Development Kit (JDDK) provides the necessary components for system users to take advantage of the Generic Java Driver. These include:

- **JavaDriver API:**

This high-level API permits an application written in Java to exchange messages with the engine by a set of usage patterns. Javadoc is included as part of the documentation for the API.

Built on a lower-level (JNI) interface with the system engine that users can also access. The source code of the high-level JavaDriver API is included as an example of usage of the low-level API.
- **Run time help tool:**

This is in the system IDE. It generates configuration artifacts necessary for the underlying operation of the Generic Java Driver in the system engine.

This includes `Netconfig`, `.pni`, and `.ini` files.
- **System sample site:**

This includes various threads that show the message patterns usage of the JavaDriver API.

The sample site contains the artifacts that are already built and deployed, such as the Java JAR files, `.pni`, `.ini`, and `NetConfig` files.

The sample site resides under the system install location and can be installed as a real system site using the standard `hcisiteinit` command and copying the files from the sample folder.

This topic, along with the companion video tutorials, describe how to run through the sample scenarios. It also explains how the sample Java applications interact with the engine using the API to achieve the usage patterns.
- **Java sample applications:**

These have source code contained in Eclipse projects. They are the Java samples that were built and deployed into the sample system site. These Eclipse projects reside under the system install locations alongside the sample site.

Both the system sample site and the Java Eclipse projects that are described in this topic reside under the system install location. This is found in a directory named `JavaDriverSamples`.

To make the sample site operational, use the standard `hcisiteinit` command. This creates an empty site and copies the contents from the `samplesite` directory to that empty site.

Java driver debug example

Use these steps in debugging the Java driver using Eclipse. Cloverleaf Integration Services (CIS) and JDK must be installed. For best results, ensure JDK is the same version as JVM in CIS.

Use these steps in debugging the Java driver using Eclipse. Cloverleaf Integration Services (CIS) and JDK must be installed. For best results, ensure JDK is the same version as JVM in Cloverleaf.

For 64-bit Cloverleaf, use the 64-bit Eclipse with 64-bit JDK.

References include:

- Cloverleaf Java Driver documentation:
See [JDDK](#).
- JavaDriver API documentation:
[\\$HCIR00T/docs/CloverleafJavaDriverAPI/index.html](#)
- Java debug wire protocol (jdpw):
<http://docs.oracle.com/javase/6/docs/technotes/guides/jpda/jdpw-spec.html>
- Eclipse 32-bit/64-bit package:
<http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/keplersr1>

Example process

CJDBounce is used for this example. The Eclipse project and cloverleaf site are located under `$HCIR00T/JavaDriverSamples`.

- 1 Open Eclipse and create the JavaDriver project by importing the existing project from `$HCIR00T/JavaDriverSamples/eclipseprojects/CJDBounce`.
- 2 Add `JavaDriver.jar` to the project library.
 - a Open **Window > Preferences** and on the **Java > Build Path > ClassPath** Variables panel click **New**.
 - b For **Name**, specify `JAVADRIVER_LIB`.
 - c Click **File** and locate the `$HCIR00T/lib/java` folder in the sample folder structure, select the Java Driver jar, and click **OK**.
- 3 Edit and test your code as required.
- 4 Create a jdpw (Java debug wire protocol) configuration for JavaDriver testing:
In the project, select **Run > Debug Configurations**. In the **Remote Java Application Connect** tab, specify:
 - Project: CJDBounce

- Connection Type: Standard (Socket Attach)
- Host: *hostname*
- Port: 7013

5 Click **Apply**.

6 Create a `.jardesc` (`exportCJDBounceJar.jardesc`) file to build/deploy the `jar` file to the target directory.

To do this:

- Select the project, then right-click **Export**.
- In the **Export** dialog box, export a `jar` file, and click **Next**.
- In the **JAR Export** dialog box, specify the necessary information:
Select **Export generated class files and resources**.
For `jar` file, specify **CJDBounce**.
For **Options**, select **Compress the contents of the JAR file**.

7 Click **Next**.

Select **Save the description of this JAR in the workspace**, and specify the file name.

8 After the file has been created, it is only necessary to select the `.jardesc` file and click **Create JAR** from the right-click menu.

The Java Class is found in the `$HCISITEDIR/thread_name` directory where Cloverleaf loads it.

WAR files

Web app projects produce WAR files. You cannot save the script in Eclipse to create the WAR file. For the `CJDMessageserver` and `CJDRequestServer` projects in NetConfig, you must do these steps.

If you do it when the corresponding Tomcat server is running, then the effects are automatic. Tomcat notices the change and it is not necessary to restart the thread.

1 Right-click the project name and select **Export**.

2 Select **Web > WAR file** and click **Next**.

3 Click **Browse** to locate the corresponding existing WAR file.

For `MessageServer` it is `$HCISITEDIR/MessageServer/apache-tomcat-7.0.8/webapps/CJDMessageserver.war`.

4 Optionally, you can select **Optimize for a specific runtime**.

5 Click **Finish** to export the WAR file.

It is better to do this when the corresponding Tomcat instance is already running. If not, then clear the created directory under Tomcat's `webapps` and work directories for the changes to take effect when you start Tomcat. This is standard Tomcat behavior.

6 Create the JavaDriver site and open the IDE.

7 Configure the Java Driver thread. For information about the sample site, see [Usage](#).

8 Add the debug information in the JVM section.

- For Cloverleaf 6.1 and later versions, create a `process_name.pni` (`bounce.pni`) file which has the same process name in the `$HCISITEDIR` directory.

- For Cloverleaf 6.0 and earlier versions, create a `thread_name.ini` file (`Bounce.ini`) which has the same thread name in the `$HCISITEDIR` directory.

- 9 Open the new file and add the line beginning with `user=-agentlib` to the JVM section. This setting should be the same as what you defined in Eclipse.

```
[JVM]
CLASSPATH = $ROOTPATH/lib/java/JavaDriver.jar # all projects need the driver
CLASSPATH=./CJDBounce.jar # Starting in start_DIR so relative ok, get .jars
user=-agentlib:jdwp=transport=dt_socket,server=y,address=7013,suspend=y
```

- `transport=dt_socket` indicates "standard(socket attach)" defined in Eclipse.
- `server=y` indicates it works as a server to wait for the debugger to be attached.
- `address = 7013` indicates it uses port 7013, which is defined in the Eclipse configuration.
- `suspend=y` indicates the JVM stays there until the debugger is attached and runtime is resumed.

These settings can also be added in the IDE in the **Process Configuration** dialog box, in the Additional JVM Options section of the **Java Driver > Java Options** tab. The settings are saved to a `.pni` file for the bounce process.

- 10 Open the Network Monitor in the IDE and start the bounce process. To ensure `jdwp` is running, you can check the process engine log that is "enable_all" in the log level.
- 11 In Eclipse, attach the debugger to JVM.
- 12 Debug the `CJDBounce` project, that is, remote Java app, after breakpoints are set.
- 13 To end process testing, stop the bounce process in Network Monitor after resuming Java source debugging in Eclipse.

Configuration

Note: This GUI implementation is intended to help you to build the java driver `.ini` file from scratch. It does not guarantee to support user-modified contents outside of the IDE tool.

A Java protocol `.ini` file for each thread that uses the Java protocol is generated and placed into the site directory.

The thread that uses the Java protocol is configured through an `.ini` file. This file is named with the same name of the thread and must be located in the thread's site directory. For example, `myJavaThread.ini` is the name of the `.ini` file, where `myJavaThread` is the thread name.

The purpose of Thread Properties is to help configure and generate the `.ini` file. A complete description of what this file contains and the meaning of each entry can be found in [.ini and .pni entries](#) on page 1044.

External interfaces

To enable you to implement your own functionality with the Java Driver, a `CloverleafDriver.jar` Java library is provided with these methods:

- `CloverleafLink.isLinked`
- `CloverleafLink.doPassword`

- `CloverleafLink.setDriverStatus`
- `CloverleafLink.getConnection`
- `CloverleafLink.doRequest`
- `CloverleafLink.doMessagesIn`

Sending messages to the system engine

To send messages into the system engine, Java code can invoke the `CloverleafLink.doMessageIn()` method.

```
public String doMessagesIn(ArrayList<String[]> msgs, boolean doRecoveryDB)
```

- `msgs` parameter is a list of six items in a string array.
- `doRecoveryDB` is a boolean parameter that tells the engine you are putting the message into the recovery database.

This method returns a string, where OK means success or ERROR indicates failure.

This code fragment shows how to invoke the method:

```
ArrayList<String[]> callAl = new ArrayList<String[]>();
String[] retStrings = new String[DrCon.IB_SIZE]; // First message
retStrings[DrCon.IB_USER_DATA] = userDataVals;
retStrings[DrCon.IB_MSG_TYPE] = DrCon.TYPE_DATA;
retStrings[DrCon.IB_MSG] = msgS;
retStrings[DrCon.IB_TRXID] = trxidVals;
// Assemble the key/subkey into driver control for next protocol thread
String drCtl = "MSG_KEY " + keyS; // Key must exist for us
if ((subKeyS != null) && (subKeyS.length() > 0))
    drCtl = drCtl + " MSG_SUBKEY " + subKeyS;
if (expectReplyB) { // Add our thread id so can get the reply
    drCtl = drCtl + " " + DrCon.CON_DRCTL_KEY + threadIdL;
}
retStrings[DrCon.IB_DRIVER_CTL] = drCtl;
callAl.add(retStrings); // Only one message
// Call and do not wait for msg to be in rdb
String retS = cl.doMessagesIn(callAl, false);
if (retS == null) {
    return "Driver call returned null String.";
}
```

Retrieving messages from the system engine

To retrieve messages from the system engine, Java code can invoke the `CloverleafLink.doRequest()` method().

```
public ArrayList<String[]> doRequest(String[] request)
```

- Parameter `request` is five items in a string array.
- An `ArrayList` of messages is returned contained in a `String[]`.

This code fragment shows how to invoke the method:

```
ArrayList<String[]> callAry = new String[DrCon.RQ_PARA_LEN];
callAry[DrCon.RQ_KEY] = keyS;
callAry[DrCon.RQ_JAVA_OP] = operationS;
```

```
callAry[DrCon.RQ_SUB_KEY] = (subKeyS.length() == 0) ? null: subKeyS;
callAry[DrCon.RQ_INDEX] = (indexS.length() == 0) ? null: indexS;
callAry[DrCon.RQ_NAME] = (nameS.length() == 0) ? null: nameS;
retAl = cl.doRequest(callAry);
```

Environment variables defined in the Generic Java Driver that can be used in Java code

The Generic Java Driver always includes these "defines" for the JVM. They are available in Java at any time through the system properties as used with environment variables:

- *THREADNAME*: Driver thread name.
- *ROOTPATH*: Install directory path.
- *SITEPATH*: Site directory path.
- *PROCESSPATH*: Thread process directory path.
- *PROCESSNAME*: Running driver process name.
- *USERRECOVERY*: TRUE/FALSE from thread properties for recovery database use.

Note: On Windows, the environment variables are set at the time the engine process starts. They cannot be modified later. The JVM can only see and use what was set for the process. You must use "defines" in the configuration file instead of environment variables for all drivers sharing a JVM when the first driver starts.

Driver library

The driver library is contained in `CloverleafDriver.jar`, which is located in the `$HCIR00T/lib/java/` folder. This must be put on the classpath in the `.ini` file for all Java driver protocol threads.

Release compatibility

Earlier NetConfigs can be read by newer versions of the software.

Later NetConfigs that contain configuration information from the preceding sections are not compatible with earlier versions of the software.

Thread properties

Java configuration is performed on the Network Configurator's **Thread Configuration** dialog box. The Java protocol is available on all platforms. Selecting it from the list indicates that the thread uses the Java class to transfer data.

The engine ignores the `4Encoding` setting in the NetConfig file and leave the Java code to handle encoding. The **Encoding** list is disabled on the Network Configurator when you configure a Java protocol thread.

Java Driver Protocol Properties dialog box

Selecting the Java protocol and then clicking **Properties** opens the **Java Driver Protocol Properties** dialog box.

Type has two options:

- Class (default)
- Application

This corresponds to the DRIVERTYPE property in the DRIVER section.

The Class text box shows the class name of the Java driver. It corresponds to the CLASS property in the DRIVER section. This text box is read-only. Click **Browse** beside the text box to select a valid class from Class Chooser dialog box.

A valid class is one that implements any required driver methods according to the naming convention. For example, if the driver type is Application, the doStart and doStop methods are required and must be implemented in the class.

If the doMsg and doReply methods both are implemented in the selected class, then a dialog box opens. In this dialog box, you can determine which one should be the Run method of the Java driver.

After a class is selected, the jar file or the directory that contains the class is automatically be added to the Classpath.

Using LinkClass

When a custom implementation of ToCloverleafLink is used, LINKCLASS is overridden in the *thread.ini* driver's config file and NetConfig is saved. When this happens, the LINKCLASS attribute is replaced with the ToCloverleafLink class.

To override this, select a Link Class attribute to maintain the LINKCLASS.

Class Chooser dialog box

This dialog box permits you to select a Java class under the root directory. When a jar file is clicked in the dialog box, the dialog box shows the content of the jar. At this point, you select a class from the jar file.

This dialog box validates whether the selected class is valid or not when **OK** is clicked. If the class is valid, then the class name is shown in the **Class** field. Otherwise, an error message opens according to the error type.

Methods

The Methods section lists all implemented driver methods of the class and the arguments of the methods. If the method has its argument option, then the corresponding argument cell is enabled; otherwise, the cell is not available. When it is enabled, you can add or modify the argument for the method.

You can use the check box column to select/clear methods. Unchecked methods are not written into the thread .ini file. When doTimeEvent is unchecked, the Time Method(doTimeEvent) Options panel is disabled. Otherwise, the panel is enabled and **Interval** or **Use advanced scheduling** must be selected.

Because the driver methods in the class can be re-defined after configuration, the tab has a **Refresh**. Clicking this updates the class to show the newest method and method argument list in this section.

During refreshing, the dialog box performs the same validation as when selecting a class to ensure the class is valid after being redefined.

Messages queue

Checking **Enable message queues** enables the keyed message mode for this thread. By default, the check box is unselected, and **Maximum Available Memory** and **Report error when a message without key arrives** are not available.

- **Maximum Available Memory** is used to set the maximum memory that is used for "in memory messages" in the thread at runtime. The memory range is 1 to 500 MB. If the text box value is empty, then the default memory of 150 MB is used.
- **Report error when a message without key arrives** specifies what happens if a message without a key arrives in this thread. By default, the check box is unselected and the arrival of messages without keys is considered an error.

Time method options

The Time method options are enabled when the selected class implements the TIME method.

The **Interval** field is used to configure the interval that the TIME method is periodically called when the interval expires. The interval is in seconds and can be empty.

If **Use advanced scheduling** is checked, then the **Interval** field is disabled and **Setup** beside the check box is enabled. Clicking this opens the **Scheduling** dialog box that permits you to set up an advanced scheduling time instead of an interval.

If both the interval and the advanced scheduling time are not configured, then the time scheduling is disabled.

Sleep time

This option specifies how much time the driver sleeps after the JVM starts. By default, this option is disabled.

Skip JVM shutdown hooks

This option specifies whether the JVM shutdown hooks are invoked when the thread exits. This corresponds to the NO_JVM_DESTROY property in the DRIVER section. By default, the check box is unselected.

Validation

When **OK** is clicked on the properties dialog box, the data validation is performed. If the validation fails, then the dialog box switches to the tab that has the incorrect option values and prompts an error message. Error messages are prompted when the corresponding option value is in error.

When no driver class is selected, an error dialog box opens.

If the driver type is `Application`, then the `doStart` and `doStop` methods are required. If any of the methods are not implemented in the selected class, then an error dialog box opens.

Thread template

When a thread that uses the Java protocol is saved as a thread template, the Java protocol configuration `.ini` file of the thread is copied to the template folder. This is copied as the protocol template of the thread template. This is located at `\integrator\templates\NetConfig`.

The protocol template file uses `java.ini` as a suffix to differ from the thread template file.

When a thread is created from a template that has the protocol template, the protocol template file is copied to the site directory. This directory has the same name as the new thread.

JVM auto defines

The driver always includes these defines for the JVM. They are available in Java at any time through the system properties. For example, `System.getProperty("THREADNAME")`.

- `THREADNAME`: The name of the driver thread.
- `ROOTPATH`: The path to the install directory.
- `SITEPATH`: The path to the site directory.
- `PROCESSPATH`: The path to the thread's process directory.
- `PROCESSNAME`: The name of the process in which the driver is running.
- `USERRECOVERY`: `TRUE/FALSE` from thread properties for recovery database use.

To use environment variables, it should be understood that they are set at the time the engine process starts. They cannot be later modified so the JVM can only see what was set for the process.

.ini and .pni entries

The thread that uses the Java protocol is configured through an `.ini` file with the same name of the thread. This must be located in the thread's site directory.

- The thread-level `.ini` file consists of `[DRIVER]`
This section contains the parameters for this instance of the driver.

- The process that uses the Java protocol is configured through a .pni file.
- The process level .pni file consists of [JVM]

This section sets up the JVM that is used by the driver.

The .ini and .pni

Similar to the JVM auto defines, the .ini and .pni files have some predefined variables that they can use. These are the subset of the auto defines, used by prefixing with "\$" as shown in the sample: files ignore case on keys, permit the normal line continues using a "\" and comments use the "#" character. Comments may follow a key value or be a complete line in the file. Comments cannot use the "\" to continue to the next line in the file.

- ROOTPATH: The path to the install directory.
- SITEPATH: The path to the site directory.
- PROCESSPATH: The path to the thread's process directory.

This is the content of a sample configuration file:

```
files ignore case on keys, permit the normal[DRIVER]
DRIVERTYPE = class # A class type or application type driver
CLASS = com/infor/cloverleaf/ib/IbLink # The class loaded for method calls that start/stop things
STARTMETHOD = doStart # method called when driver starts
STARTARG = # No arg required
STOPMETHOD = doStop # Do java clean up when driver stops
STOPARG = # No arg required
INITMETHOD = doInit # initialize things before the driver starts
INITARG = # Arg to init method not required
RUNMETHOD = doMsg # The method to run when the engine sends a message to the driver
RUNARG = # Run method user argument
NOTIFYMETHOD = # Class method called when new keyed message arrives
WRTOKMETHOD = # Called when engine main checks to see if ok to write message
TIMEMETHOD = # Called using timer set up by netconfig -- adv. sch. or just regular
TIMEMETHODARG = # Argument to the timemethod call
DELAY = 10 # Delay for Java debugger attach in seconds
MSG_MEM_MAX = 100 # Max memory use for keyed messages
KEYED_MSGS = FALSE # Doing keyed messages?
NO_KEY_SEND = true # True does send using RUNMETHOD, false no key is error
no_JVM_destroy = false # if false the jvm destroy method is called when jvm shuts down, if true
then it is not called
# LINKCLASS is required in cases where java sends messages to the Cloverleaf engine. It should
always be ToCloverleafLink,
# unless the user implements their own version of ToCloverleafLink.
LINKCLASS = com/infor/cloverleaf/driver/ToCloverleafLink
[JVM]
# example of how to use an endorsed directory and a define
define = java.endorsed.dirs=endorsed
classpath = $ROOTPATH/lib/java/JavaDriver.jar # all projects require the driver
classpath = ./InboundServer.jar # Starting in site so relative ok, get .jar's
classpath = . # Get any classes there and maybe the wsdl file if required
mem_min = 64
mem_max = 128 # Little required for testing
# Turn on remote debug
user = -agentlib:jdwp=transport=dt_socket,server=y,address=7011,suspend=n
# Set the working dir for the JVM as the dir specific to this thread. By convention it is the
same as the thread name, but can be somewhere else.
start_dir = $SITEPATH/InboundServer
```

The engine supports case-insensitive entry names, but the GUI does not. The GUI only supports uppercase entry names. If there is a lowercase, or mixed, name in the file, then the GUI fails to load.

This is a list of the available .ini and .pni entries, their meanings, and a list of samples that make use of them. This list is broken down into two parts:

- Driver section entries (`.ini`)
- JVM section entries (`.pni`)

All entry names are case-insensitive. Values are not necessarily case-insensitive, depending on what is using them. In this case, you should assume they are case-sensitive.

Driver section

This table shows the Driver section entries in the `.ini` and `.pni` files:

Name	Description	Sample threads
DRIVERTYPE (Required)	Permitted values are <code>class</code> or <code>application</code> . <code>class</code> means that the method defined in the <code>STARTMETHOD</code> entry is expected to run for a short time and then exit. <code>application</code> means that the <code>STARTMETHOD</code> is expected to run forever, so the driver spawns a separate thread to run it in.	All
CLASS (Required)	This defines the fully qualified class that contains all the methods mentioned in this list. All of the methods that are defined are members of this class.	All
INITMETHOD (Optional)	An optional method in the class that can be used to set up things before the driver is running. Example: <code>public int doInit(String initarg)</code> . If the return value is non-zero, then it is assumed that there was an error and the driver enters the error state.	WeatherClient
INITARG (Optional)	A string value that is passed to in the invocation to the <code>init</code> method. It is optional and can be null in the invocation, so should be checked before using in the method.	WeatherClient (blank)

Name	Description	Sample threads
STARTMETHOD	<p>Method that is invoked when all of the driver is operational. It is required for the application mode where the method is invoked from a special thread. It does not return from the i until the STOPMETHOD is invoked on shut down.</p> <p>For the class mode it must return if called.</p> <p>Example: <code>public void doStart(String startarg)</code>. Runs after any INITMETHOD.</p> <p>Optional if <code>DRIVERTYPE=class</code> Required if <code>DRIVERTYPE=application</code></p>	<p>CallbackJava Messageserver Requestserver</p>
STARTARG (Optional)	<p>The string argument to the STARTMETHOD. Check for null before using to avoid an exception.</p>	<p>CallbackJava (blank) Messageserver Requestserver</p>
STOPMETHOD	<p>Called during driver shutdown before operations in the driver are terminated.</p> <p>Example: <code>public void doStop(String stoparg)</code></p> <p>Optional if <code>DRIVERTYPE=class</code> Required if <code>DRIVERTYPE=application</code></p>	<p>CallbackJava Messageserver Requestserver</p>
STOPARG (Optional)	<p>The string argument to the STOPMETHOD. The user should check for null before using to avoid an exception.</p>	<p>CallbackJava (blank) Messageserver (blank) Requestserver (blank)</p>
NOTIFYMETHOD (Optional)	<p>Called in the keyed message mode (see <code>KEYED_MSGS</code> (optional)) when a new keyed message is placed in the driver queue.</p> <p>It has two calling parameters:</p> <ul style="list-style-type: none"> • The number of messages with the key that are in the driver queue. • The key value of the new message. <p>Example: <code>public void doNotify(int count, String msgKey)</code></p>	<p>Messageserver</p>

Name	Description	Sample threads
TIMEMETHOD	<p>If the driver is using a time in the NetConfig, then this method is called as specified by the user in the driver setup. The time that is specified can be seconds or can be from Advanced Scheduling.</p> <p>Times and Advanced Scheduling uses the main protocol thread to trigger the events. If the invocation returns a non-zero value, then the driver is placed in the error state and this method does not invoke again.</p> <p>Example: <code>public int doTimeEvent(String userArg)</code></p>	CallbackJava
TIMEMETHODARG (Optional)	<p>The string argument to the TIMEMETHOD. You should check for null before using to avoid an exception.</p>	CallbackJava
SECONDS (Optional)	<p>The interval in seconds to invoke the time method. This overrides any entry in the NetConfig.</p>	CallbackJava (commented out example)
RUNMETHOD	<p>Method that accepts outbound messages from the driver for processing in the Java code. Only one message, with its parameters, is transferred per invocation.</p> <p>The invocation contains the user data, driver control, message type DATA or REPLY, and message content. It also contains the user args that are supplied in the .ini file.</p> <p>Example: <code>private ArrayList<String[]> doMsg(String userDataS, String driverCtlS, String msgTypeS, String msgS, String userArgS)</code></p> <p>The ToCloverleafLink and FromCloverleafLink classes implement this method privately, though ToCloverleafLink calls it doReply instead of doMsg because the name is flexible.</p> <p>Required unless NO_KEY_SEND set to false.</p>	<p>Bounce</p> <p>CallbackJava</p> <p>MyCallback</p> <p>Requestserver</p> <p>WeatherClient</p>

Name	Description	Sample threads
RUNARG (Optional)	The string argument to the RUNMETHOD. The user should check for null before using to avoid an exception.	Bounce CallbackJava MyCallback Requestserver WeatherClient (blank in all)
LINKCLASS	Fully qualified name of the class that contains the built-in methods used to invoke from Java into the driver. This should always be set to <code>com/infor/cloverleaf/driver/ToCloverleafLink</code> . The program sends messages into the system engine or queries for keyed messages, unless the user has implemented their own version of <code>ToCloverleafLink</code> .	CallbackJava Messageserver Requestserver WeatherClient
WRTOKMETHOD (Optional)	Write OK method that is called when the engine asks the driver if it can accept an outbound message. If it returns non-zero, then the driver can accept a message to write outbound. If zero is returned, then the engine does write any waiting messages. This check for "ok to write" is part of all drivers. In a driver similar to the TCP Client driver, it may stop message attempts when the remote is not connected. It works the same in this driver. The default, if method is not defined, is it is "ok to write". Due to the nature of the driver connection, this can be called up to three times before a single write happens. These three calls happen only once every 10 seconds no matter how many messages are in queue. Invocations to this method do not count against a message's Outbound Retries quantity. This is good to use when the driver is down. It does not keep decrementing the Retries quantity on all messages in queue when the driver is unavailable for the time being. Example: <code>public boolean isWriteOk()</code>	Bounce

Name	Description	Sample threads
NO_KEY_SEND (Optional)	<p>In keyed message mode the value sets what happens if an outbound message with no key arrives in the driver.</p> <p>If the value is true, then the RUNMETHOD is called.</p> <p>If it is false (default), then the arrival of the message in the driver is considered a serious error.</p>	Messageserver
MSG_MEM_MAX (Optional)	<p>In keyed message mode, this sets the maximum memory use for "in memory" messages. When the limit is reached, outbound messages that are stored in the recovery database and retrieved as required.</p> <p>The value is in megabytes.</p> <p>The maximum permitted memory is 500 megabytes.</p> <p>The default is 0.</p>	Messageserver
KEYED_MSGS (optional)	<ul style="list-style-type: none"> • If the value is true, then the keyed message mode is active. • If the value is false (default), then driver control is not inspected for a key on inbound messages. Request calls to the <code>executeQueueQuery</code> method are errors. 	Messageserver
NO_JVM_DESTROY (Optional)	<p>If false (default), then when the thread exits the JVM's destroy method is called, which invokes any shutdown hooks.</p> <p>If true, then these shutdown hooks are not called.</p> <p>This exists because some applications do not seem to shut down correctly and the JVM destroy method never returns. It is better to stop the JVM by setting this "true".</p>	None

Name	Description	Sample threads
DELAY (Optional)	<p>If the value is greater than zero, then the engine/driver sleeps this number of seconds after starting the JVM. It does this before anything is called or it is ready to run.</p> <p>This is used to give time to attach a Java debugger to the JVM. Before the delay starts, a message is written to the error log so that by watching that file you know when to do the attach. (Or set it for 20 seconds, start the thread, wait about 7 seconds and attach).</p> <p>Default 0</p>	All

JVM section

This table shows the JVM section entries in the `.ini` and `.pni` files:

Name	Description	Sample threads
DEFINE (Optional)	<p>The value is passed to the JVM as <code>-Dvalue</code> to define properties.</p> <p>The <code>-D</code> is added by the driver. The number of these entries can be from zero to whatever is required. You should not try to put in your own <code>-D</code> as this can cause problems for the string arrays that are used during JVM startup.</p> <p>Multivalued</p>	All
CLASSPATH	<p>One directory or jar file that goes on the classpath. You should use one entry for each item, as the driver reformats these entries for the JVM startup invocation.</p> <p>There should be at least one of these entries in configuration. Note that paths can be relative to the working directory used by Java as defined by the <code>start_dir</code> entry.</p> <p>Supported wildcards are:</p> <ul style="list-style-type: none"> <code>./*</code> <code>*/.jar</code> <code>./*.JAR</code> <p>Required to have at least one entry.</p> <p>Multivalued</p>	All

Name	Description	Sample threads
USER (Optional)	<p>An option that does not require the <code>-D</code>. A good example is the option that turns on Java debugging for the JVM. Most of these start with the minus sign.</p> <p>One value per entry as the JVM, unlike command line starts, uses string arrays which may ignore a second value.</p> <p>Multivalued</p>	All
MEM_MIN (Optional)	<p>Sets minimum heap space memory for the JVM. The value is in megabytes.</p> <p>Default is 64</p>	All
MEM_MAX (Optional)	<p>Sets minimum heap space memory for the JVM. The value is in megabytes.</p> <p>Default is 64</p>	All
JVM_PATH (Optional)	<p>Usually the JVM library that is loaded is the one which is in the system install. If you must change the loaded JVM, for example, to a different version, then set this parameter to the directory path to the library. The form of the path depends on the OS but should end in the library file name: for example <code>libjvm.so</code> in Linux and <code>jvm.dll</code> in Windows.</p> <p>Default is JVM installed with the system.</p> <p>To use a JVM different from the default, you must set the environment variables. Then, start the thread using the system command-line commands, not the GUI.</p>	Commented out examples in <code>MessageServer</code> and <code>RequestServer</code>
START_DIR (Optional)	<p>The working directory for the JVM. This sets the System property <code>user.dir</code> to this path. Note that when a JVM is started you cannot change the working directory in the Java code.</p> <p>Best practice is to create a directory under the site with the thread's name and make this the <code>START_DIR</code>. Then, you can copy jar files and other required items, and reference <code>CLASSPATH</code> entries relative to this directory.</p> <p>Note: Not everything the JVM does seems to be at this path. Java looks for some items from the working directory and some from the path where the JVM was started.</p> <p>For example, when using the <code>define</code> field and setting <code>java.util.logging</code> properties files, it starts from the process directory instead. This is the reason the Bounce sample uses <code>\$SITEPATH</code> to locate the <code>logging.properties</code> file.</p> <p>Creating a file at <code>"."</code> creates the file in the process directory. A best practice to access this path from Java code is to always use the <code>user.dir</code> system property to locate this directory.</p> <p>Default is the current process directory of the process that contains the driver, the standard directory for threads.</p>	All

JVMPATH precondition

For JVMPATH, there is a precondition before you can use this feature, that is, you must add some paths to the environment.

- Windows:

```
set PATH =  
%JVMROOT%\bin;%JVMROOT%\bin\client;%PATH
```

The JVMROOT is the path to the JVM, for example, C:\JDK1.6.1_25\jre.

- Linux:

```
export  
LD_LIBRARY_PATH=$JVMROOT/lib/i386:$JVMROOT/lib/i386/server:$LD_LIBRARY_PATH
```

For JVMPATH to specify a JVM other than the default, you must use the command-line utilities to start the thread, not the Network Monitor.

Installed Java Driver library

The driver library is contained in `JavaDriver.jar` which is located in `$HCIRoot/lib/java/`.

This must be put on the classpath in the `.ini` file for all Java driver protocol threads. This does not apply if you have implemented your own version of the Java Driver code on the Java side.

Usage

There are two API levels:

- The lower API is the JNI interface.
- The upper API is provided in `JavaDriver.jar`, referred to hereafter as the Java Driver API. This upper API can optionally be replaced by the user with calls directly to the JNI interface.

Information in this section is based on the Java Driver API. The JNI interface is described in JNI interface.

The `JavaDriver.jar` contains the source code along with the class files, where you can debug into the source code of this API.

Usage patterns

Usage patterns that the Java Driver user can expect:

- A piece of Java code is invoked every time this thread receives a message, that is, from another system thread. The return value of the Java code is turned into a list of reply messages back into the system engine of this thread. The thick arrow indicates the possibility of zero or more messages; the thin arrow means one message. The `FromCloverleafLink` class should be extended to implement this pattern.
- This thread contains any number of named queues.
 - Messages coming from other system threads are placed into these named queues.
 - A piece of Java code can then issue various commands (queries) to access the messages in the named queues and manage the queues. Examples include: retrieving metadata about the messages, retrieving actual messages, deleting messages, and others.
 - The `CloverleafLinkAbstract` or `ToCloverleafLink` class should be extended to implement this pattern.
- A piece of Java code can create one or more messages, send them into the thread's system engine, and expect a reply message from the engine.

The processing of the reply can take two forms:

- **Await reply:** The Java code that submits the messages into the engine waits for the reply to come back and process it. Extend the `ToCloverleafLink` class to implement this pattern.
- **Callback:** A callback is defined to process the reply when it comes back from the engine. The Java code that submits the message can do other tasks. Extend the `ToCloverleafLink` class to implement this pattern.

Or, the Java code can not process any reply. The `CloverleafLinkAbstract` or `ToCloverleafLink` class should be extended to implement this pattern.

Patterns involving Java initiating messages (3.a, 3.b) or queries (2.b) require a Java thread to be started. This sends messages into the engine based on some stimulus, such as starting a web server that sends messages into the engine to answer requests. Patterns that react to requests from the engine can be a Java library that is passively invoked.

Javadoc for Java Driver API

Refer to the Java doc HTML pages for detailed descriptions of the classes in the API, such as the aforementioned `FromCloverleafLink`, `ToCloverleafLink`, or `CloverleafLinkAbstract` classes.

These pages are located at `$HCIR00T/docs/CloverleafJavaDriverAPI/index.html`.

Sample applications

The sample applications, system sites with Java protocol threads and associated Java code, show the usage of the API to implement the patterns previously described.

Video tutorials are available that demonstrate the sample applications:

- **Samples Walkthrough:** A general overview of what the samples cover, their structure, and message flow.
- **Using the Samples:** Shows the steps of starting up processes, Looks at the logs to see what happened. How to use a web browser to interact with the web portions of the samples.

- **Samples Debugging:** Demonstrates using Eclipse to debug the Java code across different sample threads in real time when running it in the system.

These videos can be found at the Infor Support Portal or Concierge.

WeatherClient

This demonstrates pattern 1.

See [Bouncefile and Bounce \(pattern 1\)](#).

WeatherClient is a Java thread that receives a message from the system engine by the Java Driver API. Then, it calls an external web service to get actual weather information and reply.

Bouncefile and Bounce (pattern 1)

Bouncefile is a regular system file thread that picks up a file and routes it to Bounce. This is a Java thread that invokes a piece of Java code to process the message and reply. It then gets routed back to Bouncefile with results appended to an output file.

See the *Samples Walkthrough* video for a more detailed depiction of message flow.

This Bounce.ini shows how it is configured:

```
[DRIVER]
DRIVERTYPE = class # A class type or application type driver
CLASS = com/infor/cloverleaf/javadriver/samples/cloverleafmessagebounce/Bounce
# The class loaded for method calls that start/stop things
RUNMETHOD = doMsg # all classes that override FromCloverleafLink must set RUNMETHOD to doMsg
RUNARG = # Run method user argument
STARTMETHOD = doStart
DELAY = 0 # Delay for Java debugger attach in seconds
WRTOKMETHOD = isWriteOk # Called when engine main checks to see if ok to wrt message.....
```

In this example:

- The CLASS entry shows where the Java code is located that processes the message from the system engine in Bounce.
- The DRIVERTYPE (=class) indicates that the doStart() method in the class returns. A DRIVERTYPE (=application) indicates the method does not return.

Additionally, the Bounce sample demonstrates the WRTOKMETHOD field in the .ini file. In the Java source code the matching isWriteOk method writes a log entry and returns true.

This is a segment of the Java code in Bounce.java, which splits the incoming message and builds many reply messages.

```
package com.infor.cloverleaf.javadriver.samples.cloverleafmessagebounce;
...
public class Bounce extends FromCloverleafLink {
    private static final Logger logger = Logger.getLogger(Bounce.class.getName());
    private boolean firstRun = true;
```

```

@Override
public List<ToCloverleafMessage> processMessageFromCloverleaf(FromCloverleafMessage from
CloverleafMessage) throws BadDataException, RetryException {
    try{
        if (fromCloverleafMessage.getDriverControl().contains(ToCloverleafLink.DRIVERCON
TROL_CALLBACKUNIQUE)){
            List<ToCloverleafMessage> response = new ArrayList<ToCloverleafMessage>(1);
            response.add(new ToCloverleafMessage(null,fromCloverleafMessage.getDriverCon
trol(),MessageTypeEnum.DATA, "CALLBACKJAVA",null,"bouncing back unique- " + fromCloverleafMes
sage.getMessage()));
            logger.fine("bouncing back a " + ToCloverleafLink.DRIVERCONTROL_CALLBACKUNIQUE
+ " message");
            return response;
        }else if (fromCloverleafMessage.getDriverControl().contains(ToCloverleafLink.DRIVER
CONTROL_CALLBACKREGISTERED)){
            List<ToCloverleafMessage> response = new ArrayList<ToCloverleafMessage>(1);
            response.add(new ToCloverleafMessage(null,fromCloverleafMessage.getDriverCon
trol(),MessageTypeEnum.DATA, "CALLBACKJAVA",null,"bouncing back registered- " + fromCloverleafMes
sage.getMessage()));
            logger.fine("bouncing back a " + ToCloverleafLink.DRIVERCONTROL_CALLBACKREGISTERED
+ " message");
            return response;
        }else if (fromCloverleafMessage.getDriverControl().contains("MYCALLBACK")){
            List<ToCloverleafMessage> response = new ArrayList<ToCloverleafMessage>(1);
            response.add(new ToCloverleafMessage(null,fromCloverleafMessage.getDriverCon
trol(),MessageTypeEnum.DATA, "MYCALLBACK",null,"forwarding custom callback message- " + from
CloverleafMessage.getMessage()));
            logger.fine("forwarding a MYCALLBACK message");
            return response;
        }else{
            // FILE message
            String request = fromCloverleafMessage.getMessage();
            String[] requestFields = request.split(" ", 2);
            int numberOfMessages = Integer.parseInt(requestFields[0]);
            List<ToCloverleafMessage> response = new ArrayList<ToCloverleafMessage>(num
berOfMessages);
            for (int i=0; i<numberOfMessages; i++){
                response.add(new ToCloverleafMessage(null, null, MessageTypeEnum.DATA, "FILE",
null, "bounce " + i + " " + requestFields[1]));
            }
            // test exception for use in debugger to verify retry/baddata exceptions
            boolean retry=false;
            boolean baddata=false;
            boolean ignore=false;
            if(retry) throw new RetryException("test retry exception");
            if(baddata) throw new BadDataException("test bad data exception");
            if(ignore) throw new IgnorableException("test ignorable exception");

            logger.fine("bouncing back " + numberOfMessages + " FILE message(s)");
            return response;
        }
    }catch (RetryException e){
        // just rethrow these ones
        throw e;
    }catch (IgnorableException e){
        // just rethrow these ones
        throw e;
    }catch (Exception e){
        // we'll assume any exception (BadData or otherwise) here is due to bad data since
        this is a very simple procedure
        throw new BadDataException("caught exception in processMessageFromCloverleaf: " +
e.getLocalizedMessage(), e);
    }
}

```

This sample code:

- Defines a class is defined that extends FromCloverleafLink.
- Overrides the processMessageFromCloverleaf() method, which has a parameter, FromCloverleafMessage, that represents the message object from the system engine in this thread.

- Calls `getMessage()` to get the String representation of the message.
- Formulates a list of `ToCloverleafMessage` objects as a reply back into the engine.
- Optionally throws `RetryException` to make the system engine resend the message.
- Optionally throws `IgnorableException` to indicate to ignore the error and treat the message as sent successfully.
- Optionally throws `BadDataException` to indicate an error and routes the message to the error database.
- Does "fine" level logging, which by default does not show up in system logs. See [Logging in Java Driver](#) for how to configure the logger to show fine level logging in the log files.

This sample splits the incoming message and builds many reply messages. For example, if the input message (a file dropped at the Bouncefile thread) has this content:

The number at the start of this line indicates how many messages the bouncejava thread should bounce back.

Then, the reply messages back to the Bouncefile thread is appended to the output file.

RequestServer (pattern 3.c)

This is a Java thread with a Tomcat web server. It presents to a web browser a web form, jsp page, to which you can submit pieces of data and act on them. Depending on the action, this can:

- Leave a message, without waiting for reply (pattern 3.c) with the named queue in `MessageServer` (pattern 2.a).
- Send, and wait for reply (pattern 3.a), a message to `WeatherClient`.
- Send, by a flavor of callback: unique callback (pattern 3.b), a message to `WeatherClient`.

`RequestServer` is a Java thread that has various functions. Among them, it leaves a message, pattern 3c, with the named queue in another `MessageServer` Java thread. It does this by constructing the message, complete with the driver control fields that tell the `MessageServer` receiving thread the named queue, message key, sub-key, and so on.

See the Samples "Walkthrough" video for a more detailed depiction of message flow.

As far as the receiving thread is concerned, it does not matter if the sending thread uses Java. It can be another regular system thread, as long as the necessary driver control fields are present in the message.

In this example, `LeaveAMessage.java` shows that with Java you can build and send such a message to the named queue in the receiving thread. Note the API used, `sendMessageToCloverleaf()`, which does not require to wait for the reply.

```
package com.infor.cloverleaf.javadriversamples.requestserver;
... (REMOVED FOR BREVITY) ...
public class LeaveAMessage extends HttpServlet {
    ...
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        // make sure we at least have an alphanumeric key and a message
        String messageContent = request.getParameter("MSG_CONTENT");
        String messageKey = request.getParameter("MSG_KEY");
        String messageSubKey = request.getParameter("MSG_SUBKEY");
        if (!messageKey.matches("[a-zA-Z0-9]+"))
            throw new ServletException("The message key must exist and must be alphanumeric only
with no spaces. You provided '" + messageKey + "'.");
```

```

        if (messageContent.equals(""))
            throw new ServletException("No one is interested in a blank message. Please provide
some sort of message content.");
        // for the rest, just check if they exist that they don't have spaces or funny characters

        if (!messageSubKey.matches("[a-zA-Z0-9]*"))
            throw new ServletException("Sorry, only alphanumeric values are accepted for the
subkey");
        String driverControl = "{MSG_KEY " + messageKey + "}" +
            (messageSubKey.equals("") ? "" : " {MSG_SUBKEY " + messageSubKey + "}");
        // create the message
        try {
            ToCloverleafMessage toCloverleafMessage = new ToCloverleafMessage(request.getParame
ter("USER_DATA"), driverControl, MessageTypeEnum.DATA, "LEAVE_A_MESSAGE", null, messageContent);

            ToCloverleafLink.sendMessageToCloverleaf(true, toCloverleafMessage);
        } catch (Exception ex) {
            throw new ServletException ("failed to either create the message or send it to
Cloverleaf", ex);
        }
        ...
    } finally {
        out.close();
    }
}

```

This code is an `HttpServlet`. It is a part of a web application that collects information using a web form from user input in a browser. The web application is deployed on a Tomcat web server. This is a Java application that runs independently, opposed to one that is passively invoked. Therefore, in the `.ini` file for this thread, you must indicate the method name in the `STARTMETHOD` entry to invoke to start the independent application.

Another useful point is that the driver control is formatted in the style of a Tcl keyed list. This is the practice of system drivers to format driver control this way. You should create your entries in this way, to be compatible with any driver control data added by `ToCloverleafLink` class.

See the `ToCloverleafLink` class in the Java doc for more information on driver control.

This is an example of `RequestServer.ini`:

```

[DRIVER]
DRIVERTYPE = class # A class type or application type driver
CLASS=com/infor/cloverleaf/javadriver/samples/requestservertomcatlauncher/TomcatLauncher # The
class loaded for method calls that start/stop things
STARTMETHOD = doStart # this method starts tomcat
STARTARG= apache-tomcat-7.0.8 # this specifies the tomcat folder (catalina home). It is relative
to the start_DIR parameter, or it can be an absolute path.
STOPMETHOD = doStop # Do java clean up
STOPARG= # No arg required
RUNMETHOD = doReply # The outbound reply message if weather inbound
RUNARG= # Run method user argument

```

You then define the class `TomcatLauncher` with the methods `doStart()` and `doStop()`.

The invocation to `bootstrap.start()` is the standard way to start the Tomcat server in Java. It is not related to any of the Java Driver API that is discussed in this section.

```

package com.infor.cloverleaf.javadriver.samples.requestservertomcatlauncher;
...
public class TomcatLauncher extends CloverleafLinkAbstract{
    private static final Logger logger = Logger.getLogger(TomcatLauncher.class.getName());
    private Bootstrap bootstrap;
    @Override
    public void doStart(String userS) throws Exception{
        try {
            // if the user supplied string starts with "/" or a drive letter then they're

```



```

        // specifying the absolute path, otherwise assume it's a relative path
        String catalinaHome;
        if (userS.startsWith("/") || userS.matches("[a-zA-Z]:.*"))
            catalinaHome = userS;
        else
            catalinaHome = System.getProperty("user.dir") + File.separator + userS;
        // check if the file exists at least
        File catalinaHomeFile = new File(catalinaHome);
        if (!catalinaHomeFile.exists()){
            throw new Exception("catalina home '" + catalinaHome + "' is not found, cannot
start tomcat without this. Check your STARTARG.");
        }
        // set the catalina.home property that bootstrap relies on
        System.getProperties().put("catalina.home", catalinaHome);
        // fire up tomcat
        bootstrap = new Bootstrap();
        bootstrap.start();
        // log it
        logger.info("started tomcat at: " + userS);
    } catch (Exception ex) {
        // we failed to start tomcat, throw an exception so the thread will change to error
state
        throw new RuntimeException(ex);
    }
}

@Override
public void doStop(String userS) {
    try {
        // stop tomcat gracefully
        bootstrap.stop();
        logger.info("stopped tomcat successfully");
    } catch (Exception ex) {
        logger.log(Level.SEVERE, "failed to stop tomcat properly", ex);
    }
}
}

```

MessageServer (pattern 2.a)

This is also a Java thread with a Tomcat web server. It can present a web form and take in data from the web browser. Then, it queries or perform various operations on the named queues in this thread, pattern 2.b.

By being a Java thread with the correct configuration, it can receive messages coming from other system threads into its named queues.

See the Samples "Walkthrough" video for a more detailed depiction of message flow.

The .ini file example specifies that this Java thread has `KEYED_MSGS = true`, so that named queues are enabled. RequestServer builds a message with driver control fields `MSG_KEY` and `MSG_SUBKEY`. This indicates the named queue and the sub-key of the message.

This is an example of `MessageServer.ini`:

```

[DRIVER]
DRIVERTYPE = class # A class type or application type driver
CLASS = com/infor/cloverleaf/javadriver/samples/messageservertomcatlauncher/TomcatLauncher # The
class loaded for method calls that start/stop things
STARTMETHOD = doStart # this method starts tomcat
STARTARG = apache-tomcat-7.0.8 # this specifies the tomcat folder (catalina home). It is relative
to the start_DIR parameter, or it can be an absolute path.
STOPMETHOD = doStop # stops tomcat
STOPARG = # No arg needed
DELAY = 0 # Delay for Java debugger attach in seconds
MSG_MEM_MAX = 100 # Max memory use for keyed messages

```

```
KEYED_MSGS = true # Doing keyed messages?
NO_KEY_SEND = false # True does send using RUNMETHOD, false no key is error
NOTIFYMETHOD = doNotify # Class method called when new keyed message arrives
```

This sample also demonstrates the `NOTIFYMETHOD` field in the `.ini` file, with the corresponding `doNotify` method. The method only logs the notification. Every time a new keyed message comes in, the `NOTIFYMETHOD` is called.

MessageServer (pattern 2.b)

This Java thread shows how you can access the contents of the named queue by running specific operations in a query. In general, you use the Java Driver API to access the names queues in its own thread.

`MessageServer` starts its own independent Java application. This is a Tomcat web server as in `RequestServer`, that listens on http traffic from an external client such as a web browser.

In this sample code from `MessageServer.java`, `processRequest()`:

- Collects the browser user input regarding what type of operation is to be performed against what named queue, and so on.
- Builds up a `QueueQuery` object with the user input.
- Calls the `executeQueueQuery()` static method to run the operation on the queue.
- Formats the response from the query and sends it back to the browser.

```
package com.infor.cloverleaf.javadriversamples.messageserver;
...
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    // if we can't process the request, just toss out an exception
    // first, create a query from the request
    String indexString = request.getParameter("MSG_INDEX");
    Integer index = indexString.equals("") ? null : Integer.parseInt(indexString);
    QueueQuery query = new QueueQuery(request.getParameter("MSG_KEY"), request.getParameter("MSG_SUB_KEY"), request.getParameter("MSG_NAME"), index, QueueQuery.OperationEnum.valueOf(request.getParameter("MSG_OPERATION")));
    List<QueueQueryMessage> queryResponse;
    try {
        queryResponse = ToCloverleafLink.executeQueueQuery(query);
    } catch (Exception ex) {
        throw new ServletException(ex);
    }
    response.setContentType("text/html;charset=UTF-8");
    ...
}
public static String formatQueryResponse(QueueQuery queueQuery, List<QueueQueryMessage> response){
    String retS;
    retS = "+++++ Request Parameters +++++<br/>";
    retS = retS + "    Request Key: " +
        printString(queueQuery.getKey()) +
        "    Request SubKey: " +
        printString(queueQuery.getSubKey()) +
        "    Request Name: " +
        printString(queueQuery.getName()) +
        "    Request Index: " +
        printString(" " + queueQuery.getIndex()) +
        "    Request Operation: " +
        printString(queueQuery.getOperation().name()) + "<br/>";
    // Process the return list of items that match request
    if (response.isEmpty()) {
        return retS + "<br/>Request returned zero length ArrayList.<br/>";
    }
}
```

```

String itemS = (response.size() == 1) ? " item" : " items";
retS = retS + "+++++ Request returned " + response.size() + itemS + ". +++++<br/>";
for (QueueQueryMessage message : response){
    retS = retS + " Message<br/>" +
        " Message Key: " + printString(message.getKey()) +
        " Message SubKey: " + printString(message.getSubKey()) +
        " Message Name: " + printString(message.getName()) +
        " Message Time: " + printString(message.getTime()) +
        " Message Index: " + printString(" " + message.getIndex()) +
        " Message Length: " + printString(" " + message.getLength()) +
        " Message User Data: " + printString(message.getUserData()) +
        " Message Driver Control: " + printString(message.getDriverControl()) +
        " Message Type: " + printString(message.getType().name()) +
        " Message: " + printString(message.getMessage()) +
        "<br/>";
} // End loop over input array list
return retS;
}
...

```

RequestServer (pattern 3.a)

This has another servlet that takes browser input by asking for weather information. Then, it uses pattern 3.a to send the message into the system engine. This routes the message to another Java thread, WeatherClient. This pattern has the Java code in RequestServer send the message into the engine and wait for reply. The WeatherClient thread creates the weather information, routes it back, and then sends a message back to MessageServer to store a copy there.

Note the use of the API `sendToCloverleafAndWait()` in this section from `GetWeather.java`:

```

package com.infor.cloverleaf.javadriversamples.requestserver;
...
public class GetWeather extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String zipCode = request.getParameter("ZIP_CODE");
        if (!zipCode.matches("[0-9]{5}(-[0-9]{4})?"))
            throw new ServletException("The zip code field must resemble a zip code. You provided '" + zipCode + "'.");
        String driverControl = "{MSG_KEY " + zipCode + "}";
        FromCloverleafMessage weatherResponse;
        // create the message
        try {
            ToCloverleafMessage toCloverleafMessage = new ToCloverleafMessage(request.getParameter("USER_DATA"), driverControl, MessageTypeEnum.DATA, "GET_THE_WEATHER", null, zipCode);
            weatherResponse = ToCloverleafLink.sendToCloverleafAndWait(10000, toCloverleafMessage);
        } catch (Exception ex) {
            throw new ServletException ("failed to either create the message or send it to Cloverleaf", ex);
        }
    }
}

```

RequestServer (3.b Unique Callback)

In this pattern, RequestServer has a third servlet to get weather information from WeatherClient, as in the pattern 3.a above. The difference is that it uses the callback Java Driver API to achieve the same thing. The Java code that initiates the request can do other things when waiting for the reply.

In this sample code (`getWeatherUniqueCallback.java`), the Java code:

- Instantiates an anonymous class that implements the `CallbackInterface` interface.
- Passes the anonymous class as a parameter to the static `sendToCloverleafWithUniqueCallback()` method. This manages sending the message to the system engine and invokes the configured Java callback when the reply comes back from the `WeatherClient` thread.

This sample is one example of the callback mechanism, unique callback. With this, you can specify a unique instance of the callback class as the handler of this particular reply. This is opposed to a registered callback, which is described in another sample.

When the reply comes back, the `doCallback()` method of the anonymous class is invoked to construct the appropriate response back to the browser.

```
package com.infor.cloverleaf.javadriversamples.requestserver;
protected void processRequest(final HttpServletRequest request, final HttpServletResponse
response)
    throws ServletException, IOException {
    final String zipCode = request.getParameter("ZIP_CODE");
    if (!zipCode.matches("[0-9]{5}(-[0-9]{4})?"))
        throw new ServletException("The zip code field must at least look like a zip code.
You provided '" + zipCode + "'.");
    String driverControl = "{MSG_KEY " + zipCode + "}";
    FromCloverleafMessage weatherResponse;
    // keep track if the callback has done the work or not
    final StringBuffer status = new StringBuffer();
    try {
        // create the message
        ToCloverleafMessage toCloverleafMessage = new ToCloverleafMessage(request.getParame
ter("USER_DATA"), driverControl, MessageTypeEnum.DATA, "GET_THE_WEATHER", null, zipCode);
        // create a callback which is connected to the response stream
        /* Note: Being connected to a stream is a good example of when you should use a
unique callback
        * because if the JVM goes down the stream is lost anyway.
        * So it wouldn't matter if the callback class survived the JVM restart because the
stream
        * won't anyway. A registered handler would work, but if you didn't unregister each
one after use
        * then the list of registered handlers would increase with every call.
        */
        CallbackInterface callback = new CallbackInterface(){
            public List<ToCloverleafMessage> doCallback(FromCloverleafMessage fromClover
leafMessage) throws BadDataException, RetryException {
                // synchronize on the status object to avoid a tiny race condition
                synchronized (status){
                    // if the message already timed out, this message will never succeed-
                    // notify Cloverleaf of the permanent failure
                    if (status.toString().equals("done"))
                        throw new BadDataException("the timeout already elapsed for this
message, it cannot be processed anymore");
                    // otherwise continue
                    try {
                        // great, we got a callback, print the message out
                        writeResponse(response, fromCloverleafMessage, zipCode);
                        status.append("done");
                    } catch (Exception ex) {
                        // set status so user know something went wrong
                        status.append("callback caught exception: " + ex.getLocalizedMessage());
                    }
                    throw new BadDataException("failed to write out response to the HTTP
servlet response object due to exception, giving up", ex);
                } finally {
                    // wake up the request thread if it's still waiting
                    status.notifyAll();
                }
            }
        };
        // if no exception then everything went fine, no message to send back to CL
        return null;
    }
```

```

    }
    };
    // have the message and the callback handler ready, send to Cloverleaf
    ToCloverleafLink.sendToCloverleafWithUniqueCallback(callback, true, toCloverleafMes
sage);
  } catch (Exception ex) {
    throw new ServletException ("failed to either create the message or send it to
Cloverleaf", ex);
  }
  synchronized(status){
  try {
    // make the request thread wait (using the status object for communication)
    // for a reply for 20 seconds
    status.wait(20000);
  } catch (InterruptedException ex) {
    // we don't care if it was interrupted or timed out here, either way we
    // process it the same
  }
  // if we're not done already, it means the callback is too slow
  if (!status.toString().equals("done")){
    // set status so if callback does come eventually,
    // it'll know it's not supposed to bother the thread with whatever
    // it's current doing
    status.append("done");
    // throw an exception so the user gets a 500 error
    throw new ServletException("timed out waiting for a callback!");
  } // else the response is already written, nothing to do
  }
}
}
}

```

CallbackJava/Bounce (pattern 3.b Registered Callback)

CallbackJava and MyCallback, along with Bounce, demonstrate pattern 3.b.

CallbackJava is a Java thread that sends a message into the system engine. It then processes the reply using registered callback pattern 3.b, and another form of manual callback.

This sample shows the use of a registered callback. This is different from unique callback. If the thread that initiated the request and registered the callback is restarted before the reply comes back from another thread, then the handling of that reply is not lost. The unique callback does not guarantee this behavior.

See the Samples "Walkthrough" video for a more detailed depiction of message flow.

CallbackJava is a Java thread that runs its Java application independently. In this example, CallMeBack.java does not start a Tomcat. In its doStart() method:

- Instantiates two anonymous classes that implement the CallbackInterface interface.
- Registers the two instances as callbacks under the names red and blue, respectively.
- Goes into a loop to periodically wake up and send a message into the system engine with registered callbacks red and blue. These are based on an index that is incremented in each iteration of the loop. These messages are routed to the Bounce thread, which then bounces it back as a reply to the messages.

In the same loop, it also:

- Sends a message using the unique callback, whose usage is already covered in a previous sample.
- Sends a message (3.c) to show that the users can have a custom or manual callback in another thread, MyCallback.
- It also demonstrates TIMEMETHOD in the .ini. The time delay for TIMEMETHOD is specified in the NetConfig GUI. Every time that time delay expires (or the Advanced Scheduling algorithm, if enabled, fires an event),

the corresponding method is called (doTimeEvent, in this case). This sample method logs the event. There is a commented out field SECONDS which can override the time delay specified in NetConfig.

```
[DRIVER]
DRIVERTYPE = application # application means that the doStart method is not expected to return,
                        # so a separate thread is created to run it forever
CLASS = com/infor/cloverleaf/javadriver/samples/callback/CallMeBack # The class loaded for method
                        # calls that start/stop things
STARTMETHOD = doStart # Test that it calls it
STARTARG = # No arg required
STOPMETHOD = doStop # Do java clean up
STOPARG = # No arg required
RUNMETHOD = doReply # All classes that override ToCloverleafLink and expect reply messages from
                        # Cloverleaf into java must set this to doReply
RUNARG = # Run method user argument
TIMEMETHOD = doTimeEvent # method that runs every so often as configured in NetConfig
TIMEMETHODARG = myTimerArg # time method user argument
#SECONDS = 10 # uncomment this to override the time interval for TIMEMETHOD specified in NetConfig
```

Bounce is a Java thread shared with a previous sample that bounces back the messages coming from call backJava as their replies. It selectively passes messages onto another thread, MyCallback. This demonstrates that you can design your own custom, or manual, callback behavior in another thread, MyCallback.

From the API's perspective, this scenario does not involve any of the callbacks that are mentioned above. Messages are sent to the engine without waiting for the reply, pattern 3.c.

CallMeBack.java

```
package com.infor.cloverleaf.javadriver.samples.callback;
...
public class CallMeBack extends ToCloverleafLink {
    private boolean runningB = true;
    private Thread startThread = null;
    @Override
    public void doStart(String users) throws Exception{
        int idx = 0;
        // debugger test for exception handling. To use this, set a wait in the driver control
        // ini file, start the process, attach debugger
        if (idx == 1) throw new Exception("debug test");
        // this process just starts up and runs forever kicking off messages every 10 seconds
        startThread = Thread.currentThread();
        // create a "red" and "blue" callback handler
        CallbackInterface redCallback = new CallbackInterface() {
            public List<ToCloverleafMessage> doCallback(FromCloverleafMessage fromCloverleafMes
            sage) throws BadDataException, RetryException {
                System.out.println("RED callback executed at '" + (new Date()).toString() + "'and
                received message: " +
                    fromCloverleafMessage.getMessage());
                return null;
            }
        };
        CallbackInterface blueCallback = new CallbackInterface() {
            public List<ToCloverleafMessage> doCallback(FromCloverleafMessage fromCloverleafMes
            sage) throws BadDataException, RetryException {
                System.out.println("BLUE callback executed at '" + (new Date()).toString() +
                "'and received message: " +
                    fromCloverleafMessage.getMessage());
                return null;
            }
        };
        // now register them as callback handlers
        registerACallbackHandler("red", redCallback);
        registerACallbackHandler("blue", blueCallback);
    }
}
```

```

// loop and send out messages
while(runningB){
    // send the message to cloverleaf, don't use recoverydb since this is just for testing
    try{
        Thread.sleep(10000);
        // create a message to send
        idx++;
        // copy the int for the callback to remember (to demonstrate unique callback
        // A more realistic example of this need is in the CJDRequestServer where the
        // has to remember the HttpResponse object to write back on (a stream essentially)

        final int callbackInt = idx;
        // create a unique callback to invoke for just this one message
        CallbackInterface callback = new CallbackInterface() {
            public List<ToCloverleafMessage> doCallback(FromCloverleafMessage fromClover
leafMessage) throws BadDataException, RetryException {
                System.out.println("java callback " + callbackInt + " executed at " +
(new Date()).toString() + " and received message: " +
                    fromCloverleafMessage.getMessage());
                return null;
            }
        };
        ToCloverleafMessage toCloverleafMessage = new ToCloverleafMessage(null, "", Mes
sageTypeEnum.DATA, null, null, "callback message " + idx + " at " + (new Date()).toString());
        sendToCloverleafWithUniqueCallback(callback, true, toCloverleafMessage);
        /* now send a message to RED or BLUE registered callbacks, depending if idx is
even or odd
specific to the
handling), this
started
like this.
tract that
different
them as needed.
a shutdown, no
messages will be lost.
*/
        if (idx % 2 == 0){
            toCloverleafMessage = new ToCloverleafMessage(null, "", MessageTypeEnum.DATA,
null, null, "RED message " + idx + " at " + (new Date()).toString());
            sendToCloverleafWithRegisteredCallback("red", true, toCloverleafMessage);
        }else{
            toCloverleafMessage = new ToCloverleafMessage(null, "", MessageTypeEnum.DATA,
null, null, "BLUE message " + idx + " at " + (new Date()).toString());
            sendToCloverleafWithRegisteredCallback("blue", true, toCloverleafMessage);
        }
        /* finally, send a message with automated callback handling. This demonstrates
        * a manual callback where the callback is processed on another thread. The usage
        * scenarios are similar to the registered callbacks but this lets you use another
        * thread to handle the callbacks. You need to pass all the information the
        * handler will need in the driver control or message body so your thread on the
        * side will know what to do with the message. Using the asynchronous soap
        * an example again, you would pass the message id and the callback url in driver
        * here, then pull them out in the callback handling thread and use it to execute
        * response message to the waiting client.

```

```

        * Technically the code here has no idea a "callback" type of scenario is even
in play,      * as this thread just knows that it's dumping messages into the Cloverleaf engine.
    Your      * code will just process the driver control data appropriately to make it happen.
    Here      * we're just making up some arbitrary driver control data for the other side
to use. The   * bounce thread is just looking for this data in the driver control to know
where to send */
    toCloverleafMessage = new ToCloverleafMessage(null, "MYCALLBACK " + idx + " mes
sageid " + idx*17 + " url http://www.example.com:8080/callbackserver", MessageTypeEnum.DATA,
null, null, "BLUE message " + callbackInt + " at " + (new Date()).toString());
    sendMessagesToCloverleaf(true, toCloverleafMessage);
    }catch (Exception e){
        System.out.println("caught exception trying to sendToCloverleafWithCallback: "
+ e.getLocalizedMessage());
    }
}
}

```

JNI interface

The `JavaDriver.jar` classes provide a layer to make the JNI interface more object-oriented. Although every effort has been made to provide methods to address all usage scenarios, you might require something different. In these instances, you must write your own code to communicate to the JNI interface. This section applies only to those users who write this low-level code themselves.

The `JavaDriver.jar` contains the Java source code files as well, which serve as an example of how to use the JNI interface.

Calls from the driver

The system Java Driver can be configured to invoke your code by the `.ini` file. The [Driver section](#) describes the parameters and usage for these `.ini` fields:

- STARTMETHOD
- STOPMETHOD
- INITMETHOD
- RUNMETHOD
- NOTIFYMETHOD
- TIMEMETHOD
- WRTOKMETHOD

These parameters let you name the methods to whatever you require. An example method signature for each of these methods is provided in the table. The method name and parameter names are up to the developer. The parameter types and order are fixed.

For example, the table describes the STARTMETHOD example as `public void doStart(String startarg)` so this is valid as a STARTMETHOD `public void myStartMethod(String myArgValue)` as well.

Note: Currently, the GUI only supports generating the default method names, such as `doStart`, `doStop`, and so on. To use method names other than the default, you must manually update the `.ini` file.

These methods must be implemented in the class specified in the CLASS field of the .ini file. The `FromCloverLeafLink.doMsg` method and the `ToCloverLeafLink.doReply` method shows detailed usage of the RUNMETHOD interface. This includes:

- How to tell the Java Driver that you have accepted a message.
- That you require a message to retry again later.
- That you require a message to be moved to the error database.
- That you have accepted a message and require sending zero-to-many reply messages back in response.

Calls to the driver

When Java code invokes into the Java Driver, it specifies a LINKCLASS in the .ini file. The `ToCloverLeafLink` class in the provided `JavaDriver.jar` is such a class. This type of class can send messages into the Java Driver and from the system engine. It can also query the named queues for messages that are stored in the recovery database. After this, it sets the driver status, and invokes a password function implemented in C code.

The minimum a class must do for these things are these lines, found in the `ToCloverLeafLink.java` file:

```
private static int linkPtrI = 0;
private static native ArrayList<String[]> driverRequest(int pointer, String[] rqData);
private static native String driverMsgsIn(int pointer, ArrayList<String[]> ibData, int dbWaitI);
private static native void driverSetStatus(int pointer, int state, String errStr);
private static native String driverPwdEncode(int doEncode, String pwd);
```

- The `linkPtrI` is a pointer that the JNI code initializes to permit the Java code to invoke it. It must be present in your class with this name and type. It also must be passed as the first parameter to all the subsequent methods except `driverPwdEncode`. This does not have any thread context so does not require the pointer.
- Use the `driverRequest` method to run queries against the named queues. See the public `executeQueueQuery` method for how to invoke this method.
- Use the `driverMsgsIn` method to send messages into the driver. See the public `sendMessagesToCloverLeaf` method for how to invoke this method.
- Use the `driverSetStatus` method to set the status of the system thread, as it shows in NetMonitor. See the public `setDriverStatus` method for how to invoke this method.
- Use the `driverPwdEncode` method to invoke a C algorithm to encode a password and decode it. See the public `doPassword` method for how to invoke this method.

Some of these methods rely on constants declared nearby to make the array interactions simpler. You can copy these constants as well if they are suitable.

Minimizing inbound and outbound JVMs

Many different Java threads each doing a small amount of tasks have been created. Although good for illustrating distinct concepts, it creates many JVM instances which can each use a significant amount of system resources. To minimize resource consumption, you can opt to combine conceptually separate Java programs to run under one JVM.

To do this, the `doStart` method can start a few server programs. You can also combine a `MessageServer` type thread with a `RequestServer` thread. When messages are sent from a Java thread with different processing requirements, you can assign a different TRXID to each message type before it is sent into the engine. Then, using normal system routing, you can send the raw or translated message to whatever outbound thread is required.

Outbound threads can similarly be combined using driver control. Before the message reaches a multiplexing outbound thread, various driver control strings can be set on the messages. The outbound Java thread can examine the driver control to determine which class to invoke to handle the message.

For example, some messages might be destined for a web service endpoint or some messages might be going to an Enterprise Service Bus.

Development procedure

Use this process to create a new Java driver:

- 1 Start a normal Java project in your favorite IDE with a class having a main method.
- 2 Add the `JavaDriver.jar` to the library of your project.
- 3 Begin writing the methods you intend to use.
- 4 Test methods from a stub in `main()` until satisfied with testing.
- 5 Create a `.jardesc` file to build and deploy the jar file to the target directory, or the equivalent on other IDEs. This file is similar to the Eclipse sample.
- 6 Build/deploy your jar file.
- 7 Start the system process.
- 8 Attach debugger.
- 9 Run tests.
- 10 Edit code to fix bugs, attempt to hot swap your code changes.
- 11 If hot code swap is successful, then go to step 9.
- 12 If hot code swap fails, then stop the system process and go to step 6.

Logging in Java driver

In the Java driver, all text sent to the standard out and error streams are copied into the engine log file for the process that started this driver instance. Logging performed with `System.out.println()` calls go to the system engine log file.

You can use more sophisticated logging mechanisms such as `java.util.logging` or `log4j`. This results in those messages going to the system engine log, if those logging mechanisms are writing to standard out or standard error streams.

Commonly, these logging systems invoke these streams "logging to the console." This is true if you had started the JVM from a console.

Java `util` logging is the standard utility for logging information in Java, and is used extensively in the Java driver samples. JVMs have a built in `logging.properties` file which defines the default behavior of logging in different applications. These defaults are useful in many cases. For debugging, it is helpful to have a more detailed level of logging that you can turn down in production. In this case, it is convenient to define a `logging.properties` file specific to your system thread to log.

The Bounce sample thread does exactly this. In the `.ini` file it has this entry to make the JVM use a specified `logging.properties` file instead of its default:

```
define=java.util.logging.config.file=$SITEPATH/Bounce/logging.properties
```

This `logging.properties` file contains lines similar to this:

```
java.util.logging.ConsoleHandler.level = FINE
com.infor.level = FINE
```

These lines configure the console handler, which writes to the error stream, and logger levels to log at the FINE level. These use the `logger.fine()` calls in the Bounce thread's code to make it to the system logs.

For more information on Java `util` logging, see the official Java 6 documentation at this address. This contains a discussion of loggers, levels, handlers, and others, involved in the use of Java `util` logging:

<http://docs.oracle.com/javase/6/docs/technotes/guides/logging/overview.html>

Script UPoC

In addition to Tcl UPoC and Java UPoC, there is also a Script UPoC. Script UPoC is a special Java UPoC that supports these Java-based scripting languages:

- JavaScript
- Python

There are three types of UPoC:

- TPS, Tcl Procedure Stream, functions. These are procs in Tcl UPoC and class methods in Java UPoC.
- Trxid, Transaction ID determination, functions.
- Xlate actions, including pre-, post-, and invocation actions.

All types of UPoC are defined as functions or scripts, and are supported in the Script UPoC.

LANG, FILE, FUNC, and SCRIPT keys

For Script UPoC, items with keys `LANG`, `FILE`, `FUNC`, or `SCRIPT` specify scripting language, function, or script (code fragment).

This table lists these key descriptions:

Key	Description
LANG	Case-insensitive scripting language name: <ul style="list-style-type: none"> • "javascript" for JavaScript UPoC. • "python" for Python UPoC.
FILE	The file is searched for in the <code>scripts</code> subdirectories in these locations: <ul style="list-style-type: none"> • Current site • Master site • Root directories For JavaScript, the file names use extension <code>.js</code> and for Python <code>.py</code> . If not, then the extension is appended to the file name.
FUNC	Function name.
SCRIPT	Code snippet, which is specified directly as an argument.

Jar files and Script UPoC classes

The Script UPoC class files are packaged in `clscriptupoc.jar`. This is located in `$HCIR00T/lib/java/`. Jython jar file `jython-standalone-2.7.0.jar` is also located there.

ScriptUPoC classes and their usage are:

TPS function:

```
{ PROCS cljTPS }
{ ARGS {{{CLASS ScriptTPS} {LANG <language>} {FILE <file_name>} {FUNC <func_name>} {<key> <value>} ...}} }
```

TPS script:

```
{ PROCS cljTPS }
{ ARGS {{{CLASS ScriptTPS} {LANG <language>} {SCRIPT {<script lines> }} {<key> <value>} ...}} }
```

Trxid function:

```
{ PROC {cljTrxid {{{CLASS ScriptTrxid}}}} }
{ ARGS {{{LANG <language>} {FILE <file_name>} {FUNC <func_name>} {<key> <value>} ...}} }
```

Trxid script:

```
{ PROC {cljTrxid {{{CLASS ScriptTrxid}}}} }
{ ARGS {{{LANG <language>} {SCRIPT {<script lines> }} {<key> <value>} ...}} }
```

Xlate action function:

```
cljXLTStrings {XLT_STYLE SINGLE} {CLASS ScriptXLTString} {LANG <language>} {FILE <file_name>}  
{FUNC <func_name>} {<key> <value>} ...
```

Xlate action script:

```
cljXLTStrings {XLT_STYLE SINGLE} {CLASS ScriptXLTString} {LANG <language>}  
{SCRIPT { <script lines> }} {<key> <value>} ...
```

JavaScript UPoC

For JavaScript, the file names have a `.js` extension. The files are searched in the `scripts` sub-directories in the current site, master site, and root directories.

TPS

- Java class: ScriptTPS
- JavaScript function:

```
{LANG javascript}{FILE file_name} {FUNC func_name}
```

- JavaScript script:

```
{LANG javascript}{SCRIPT {script lines}}
```

TRXID

- Java class: ScriptTrxid
- JavaScript function:

```
{LANG javascript}{FILE file_name} {FUNC func_name}
```

- JavaScript script:

```
{LANG javascript}{SCRIPT {script lines}}
```

XLT

- Class: ScriptXLTString
- Datum Type: String
- Mode: Single
- JavaScript function:

```
{XLT_STYLE_SINGLE}{LANG javascript}{FILE file_name} {FUNC func_name}
```

- JavaScript script:

```
{XLT_STYLE_SINGLE}{LANG javascript}{SCRIPT {script lines}}
```

JavaScript scripting in UPoC

JavaScript and Python UPoC are supported.

- The JavaScript engine that is used is Nashorn.
- The Python interpreter that is used is Jython. This is an implementation of the Python programming language designed to run on the Java platform.

These are provided fully configured.

In the engine, `hcitptest` is supported.

Other support includes:

- Non-beta class names are automatically shown as javascript/python in the GUI.
- The scripts themselves directly migrate.
- For engine productization, there is an independent test tool that is the equivalent of `hcitcl`.
- In debugging, Python has special instructions for getting the engine stdin.
 - Step is supported.
 - Python commands print and update variables.
- For GUI productization, there is a GUI syntax editor.
- The GUI debugger is similar to the Xlate debugger, containing highlight/breakpoint/debug panels. Python has a built-in interpreter.

Python UPoC

For Python, the file names have a extension `.py` extension. The files are searched in the `scripts` subdirectories in the current site, master site, and root directories.

TPS

- Java class: ScriptTPS
- Python function:

```
{LANG python}{FILE <file_name>} {FUNC <func_name>}
```

- Python script:

```
{LANG python}{SCRIPT {<script lines>}}
```

TRXID

- Java class: ScriptTrxid
- Python function:

```
{LANG python}{FILE <file_name>} {FUNC <func_name>}
```

- Python script:

```
{LANG python}{SCRIPT {<script lines>}}
```

XLT

- Class: ScriptXLTString
- Datum Type: String
- Mode: Single
- Python function:

```
{XLT_STYLE_SINGLE}{LANG python}{FILE <file_name>} {FUNC <func_name>}
```

- Python script:

```
{XLT_STYLE_SINGLE}{LANG python}{SCRIPT {<script lines>}}
```

Calling functions defined in another file

Functions that are defined in another file can be called from UPoC scripts or functions. This helps code maintenance and reuse.

Before calling the functions, the file in which they are defined must be loaded into the engine. Function `clLoad()` is used to load a JavaScript file. `clImport()` imports identifiers that are defined in a Python module. Only the bare file name is necessary. The file search is directed in the `scripts` subdirectories of the current site, master site, and root directories.

JavaScript function `clLoad()`

Function `clLoad()` has this syntax:

```
clLoad(file_name);
```

`file_name` is a string of the bare file name without a path. The file name is used with or without the `.js` extension.

For example, function `foo()` is defined in a file named `bar.js`. This is called with:

```
clLoad("bar.js"); // or clLoad("bar");  
foo();
```

Python function `clImport()`

Function `clImport()` has this syntax:

```
clImport(module, *funcs)
```

`module` is the module name, or the bare file name that is the module name with the extension `.py`. The variadic parameter `*funcs` indicates you can give an arbitrary list of functions.

You can use function `clImport` similar to the `import` statement. For example, functions `foo_1()` and `foo_2()` are defined in a module named `bar`. In this instance, the file name is `bar.py`.

```
import bar  
bar.foo_1()
```

You can invoke `clImport` with only one argument:

```
clImport("bar") # or clImport("bar.py")  
bar.foo_1()
```

You can also use the `import` statement:

```
from bar import foo_1, foo_2  
foo_1()
```

You can invoke `clImport` with more arguments:

```
clImport("bar", "foo_1", "foo_2") # or clImport("bar.py", "foo_1", "foo_2")  
foo_1()
```

Java UPoC API

Script UPoC are special Java UPoC that support Java-based scripting languages.

- JavaScript scripts are run by Nashorn.
- Python scripts are run by Jython. This is an implementation of the Python programming language that is designed to run on the Java platform.

You can use all Java UPoC APIs in JavaScript and Python scripts.

You can import Java classes in this manner:

- JavaScript:

```
var DList = Java.type("com.quovadx.cloverleaf.upoc.DispositionList");
```


- Python:

```
from com.quovadx.cloverleaf.upoc import DispositionList
```

Script UPoC classes

Java classes are implemented to support different types of Script UPoC. In these classes, a Script UPoC engine, Nashorn for JavaScript or Jython interpreter for Python, is created.

These classes are used to load script files, invoke functions, evaluate code fragments, and so on.

Classes include:

- TPS
- Trxid
- XltCall
- XltObject
- XltObjects
- XltString
- XltStrings

TPS function

Scripting language:

- JavaScript

```
function TPS_func_name (cloverEnv, context, mode, msg, userArgs, dl)
```

- Python

```
def TPS_func_name (cloverEnv, context, mode, msg, userArgs, dl)
```

Args

- `cloverEnv`
This is a reference to the Cloverleaf run-time environment.
- `mode`
Run mode: `start` | `run` | `time` | `shutdown`
`time` is used only in the protocol read TPS.
- `msg`
This is a reference to the message being processed. This is `null` for `start` or `time` mode, unless there is more than one TPS on the stack and an earlier one has a message.
- `context`

This identifies which TPS-style UPoC is involved.

- `userArgs`

These are the user-supplied arguments.

- `dl`

The TPS disposition list:

- `CONTINUE`
- `ERROR`
- `KILL`
- `OVER`
- `PROTO`
- `SEND`
- `KILLREPLY`
- `RETRY`

Returns: None

Trxid function

Scripting language:

- JavaScript

```
function Trxid_func_name (cloverEnv, msg, userArgs)
```

- Python

```
def Trxid_func_name (cloverEnv, msg, userArgs)
```

Args

- `cloverEnv`

This is a reference to the Cloverleaf run-time environment.

- `msg`

This is a reference to the message being processed.

- `userArgs`

These are the user arguments that are attached to the Trxid Script UPoC.

This returns a transaction identification string that is used in message routing.

XltCall function

Scripting language:

- JavaScript

```
function XltCall_func_name (cloverEnv, xpm)
```

- Python

```
def XltCall_func_name (cloverEnv, xpm)
```

Args:

- `cloverEnv` is a reference to the Cloverleaf run-time environment.
- `xpm` is a reference to an XPM object. In this, any field in the input or output message is referenced.

Returns: None

XltObject function

Scripting language:

- JavaScript

```
function XltObject_func_name (cloverEnv, xpm, inVal)
```

- Python

```
def XltObject_func_name (cloverEnv, xpm, inVal)
```

Args:

- `cloverEnv`
This is a reference to the Cloverleaf run-time environment.
- `xpm`
This is a reference to an XPM object. In this, any field in the input or output message is referenced.
- `inVal`
This is the input object. This is the source field from the associated Xlate action that is represented as an object.

This returns the returned object. This is a type that is supported by the `com.quovadx.cloverleaf.upoc.Datum` class.

XltObjects function

Scripting language:

- `function XltObject_func_name (cloverEnv, xpm, inVals, outVals)`
- `def XltObject_func_name (cloverEnv, xpm, inVals, outVals)`

Args:

- `cloverEnv`
A reference to the Cloverleaf run-time environment
- `xpm`
A reference to an XPM object. In this, any field in the input or output message is referenced.
- `inVals`
The input vector. This is the source field from the associated Xlate action that is represented as `Objects`.
- `outVals`
The output vector. This consists of objects that are supported by the `com.quovadx.cloverleaf.upoc.Datum` class.

Returns: None

XltString function

Scripting language:

- JavaScript

```
function XltString_func_name (cloverEnv, xpm, inVal)
```

- Python

```
def XltString_func_name (cloverEnv, xpm, inVal)
```

Args:

- `cloverEnv`
This is a reference to the Cloverleaf run-time environment.
- `xpm`
This is a reference to an XPM object. In this, any field in the input or output message is referenced.
- `inVal`
This is the input string. This is the source field from the associated Xlate action that is represented as `String`.

This returns the returned object. This consists of an object with a `toString()` method that provides the representation that is stored in the destination field.

XltStrings function

Scripting language:

- JavaScript

```
function XltStrings_func_name (cloverEnv, xpm, inVals, outVals)
```

- Python

```
def XltStrings_func_name (cloverEnv, xpm, inVals, outVals)
```

Args:

- `cloverEnv`
This is a reference to the Cloverleaf run-time environment.
- `xpm`
This is a reference to an XPM object. In this, any field in the input or output message is referenced.
- `inVals`
This is the input vector. This is the source fields from the associated Xlate action that is represented as `Strings`.
- `outVals`
This is the output vector. This consists of objects with `toString()` methods that provide the representation that is stored in the destination fields.

Returns: None

Working with GitHub in Cloverleaf application development

GitHub is a popular, heavily used code repository.

This is useful for those who require source code control with merge/branch/and so on.

These topics describe best practices when using Cloverleaf application development and customization with GitHub support in healthcare IT DevOps.

Note: The Cloverleaf application user, administrator, or developer must be qualified with SCM/GitHub operations or administrations.

Preparation

Before starting, ensure these preparations have been made:

- Cloverleaf is installed.
- Git and Git client toolsets are installed.
- GitHub account and access are set.
- GitHub repository is available for Cloverleaf application site development.

Note: Git-flow is not covered in this topic.

After making the preparations, follow the steps in these topics (in this order):

- [Developing Cloverleaf with GitHub](#) on page 1080
- [Developing the Cloverleaf site with GitHub](#) on page 1082
- [GitHub reference: gitattributes](#) on page 1083
- [GitHub reference: Site-level gitignore](#) on page 1084
- [GitHub reference: Root-level gitignore](#) on page 1085

Developing Cloverleaf with GitHub

This is a demo with limited `git` commands in GitHub context. This shows Cloverleaf application development with GitHub to initialize GitHub-oriented Cloverleaf application development.

- 1 Move to the Cloverleaf root environment, and run `git init`.

```
C:\cloverleaf\cis20.1P\integrator>setroot
No default site -- no site set
C:\cloverleaf\cis20.1P\integrator>setsite helloworld

C:\cloverleaf\cis20.1P\integrator>git init
Initialized empty Git repository in C:/cloverleaf/cis20.1P/integrator/.git/
```

- 2 Use `hcigitinit` in `$HCIROOT/bin` as the helper to generate the root-level `.gitignore`, `.gitattributes`, and site-level `.gitignore` for GitHub. A context of these files is generated by `hcigitinit` and is placed in these files.

```
C:\cloverleaf\cis20.1P\integrator\sbin>hcigitinit -h
hcigitinit [options]
Options:
  --root|-r      Cloverleaf rootpath, e.g. /opt/cloverleaf/cis20.1/integrator
  --site|-s      Cloverleaf sitename(s), e.g. helloworld,testprod
  --help|-h      Help
Notes:
  To populate Cloverleaf Root and Site(s) GitHub .gitignore and .gitattributes
  if option root is not specified, try $ENV{'HCIROOT'} instead, if option site(s)
  is not specified, will try $ENV{'HCISITE'}; Otherwise it does nothing and exits.
```

```
C:\cloverleaf\cis20.1P\integrator
C:\cloverleaf\cis20.1P\integrator\sbin>showroot
HCI root is C:\cloverleaf\cis20.1P\integrator
HCI site is helloworld
C:\cloverleaf\cis20.1P\integrator\sbin>hcigitinit
### Working for the root
### C:\cloverleaf\cis20.1P\integrator
### and site(s)
### helloworld
*** C:\cloverleaf\cis20.1P\integrator\.gitignore already exists, being archived
### Generating C:\cloverleaf\cis20.1P\integrator\.gitignore
*** C:\cloverleaf\cis20.1P\integrator\.gitattributes already exists, being archived
### Generating C:\cloverleaf\cis20.1P\integrator\.gitattributes
### Appending helloworld to C:\cloverleaf\cis20.1P\integrator\.gitignore
*** C:\cloverleaf\cis20.1P\integrator\helloworld\.gitignore already exists, being archived
### Generating C:\cloverleaf\cis20.1P\integrator\helloworld\.gitignore
```

- 3 Windows EOL-style Cloverleaf configurations bring the problem to non-Windows Cloverleaf application runtimes. To ease this gap, use `git config` to set `core.autocrlf` to "true" on Windows.

```
git config --global core.autocrlf true
```

- 4 In the integrator directory:
 - a Add the artifacts using `git add`.
 - b Show the add status using `git status`.
 - c Use `git commit` to the local repository.

```
C:\cloverleaf\cis20.1P\integrator>git add .
:
C:\cloverleaf\cis20.1P\integrator>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
  new file:   .gitattributes
```

```

new file:   .gitignore
new file:   Alerts/sample.alrt
new file:   AppDefaults/GenericApp
. . . .

```

```

C:\cloverleaf\cis20.1P\integrator>git commit -m "init root and sites for github practice"
[master (root-commit) a7b9758] init root and sites for github practice
700 files changed, 89989 insertions(+)
create mode 100644 .gitattributes
create mode 100644 .gitignore
create mode 100644 Alerts/sample.alrt
create mode 100644 AppDefaults/GenericApp
. . . .

```

5 Run git remote add and git push.

```

C:\cloverleaf\cis20.1P\integrator>git remote add origin git@github.com:youracut/yourgithubrepo.git

C:\cloverleaf\cis20.1P\integrator>git push -u origin master
:
remote: Resolving deltas: 100% (65/65), done.
To github.com:youracut/yourgithubrepo.git
 * [new branch]      master -> master

```

You can also use git branch and git push after git remote add.

```

C:\cloverleaf\cis20.1P\integrator>git remote add origin git@github.com:youracut/yourgithubrepo.git

C:\cloverleaf\cis20.1P\integrator>git checkout -b dev
Switched to a new branch 'dev'

C:\cloverleaf\cis20.1P\integrator>git branch -a
* dev
  master
remotes/origin/dev
remotes/origin/master

C:\cloverleaf\cis20.1P\integrator>git push
fatal: The current branch dev has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin dev

C:\cloverleaf\cis20.1P\integrator>git push --set-upstream origin dev
Everything up-to-date
Branch 'dev' set up to track remote branch 'dev' from 'origin'.

```

Developing the Cloverleaf site with GitHub

Cloverleaf application development in a site is an incremental process. Some files and directories come in, and some move out. The git-add, git-rm, and other git operations are a routine job over the development cycle.

Working in the local repository

Use `git-checkout` to check out the base site from GitHub. After this, you can continue with Cloverleaf site development, or work with other peer's changes by resolving the version conflicts upon the `git-merge`.

Note: `gitignore` needs updating if a new file and directory are appended to the root or site for SCM management.

Teamwork by GitHub "pull" request

You can `git-commit` and `git-push` the changes to the GitHub `dev-developer` branch after unit testing has finished.

You can also make a pull request from `dev-developer` to `dev` branch.

Peer reviews are made on NetConfig or other artifacts upon pull request, and then closed by `merge`.

You can trigger DevOps CI/CD to do the checking-out or exporting (`git-archive`) of the GitHub `dev` branch for the integration deployment testing.

After a release candidate version is tested and approved on the `dev` branch, you can do a pull request from `dev` to `master`. This closes the version development on the root/site

GitHub reference: gitattributes

Root-level `.gitattributes`:

```
#Generated by hcigitinit, 2020-09-20-14-05-22
#Cloverleaf .gitattributes
#The content is encouraged to be constantly refreshed to
#reflect the needs from Cloverleaf application development
#-----

*                text=auto
rootInfo         text
siteInfo         text
siteSecurityInfo text
NetConfig        text
*.alrt           text
.alrtindex       text
.upocindex       text
*.mvw            text
*.ini            text
*.pni            text
*.idx            text
*.vrl            text
*.hrl            text
*.frl            text
*.vm             text
*.ini            text
*.policy         text
*.security       text
*.xml            text
*.xsd            text
*.xsl            text
*.xslt           text
*.tbl            text
*.xlt            text
tclIndex         text
```

```

*.tcl          text
*.pl           text
*.htc          text
*.htk          text
*.tlib         text
*.tndx         text
*.bat          text
*.log          text
*.py           text
*.js           text
*.stxdb        binary
*.db           binary
*.java         text
*.class        binary
*.jar          binary
*.war          binary
*.zip          binary
*.pdl          text
*.pdo          binary
*.der          binary
*.jks          binary
*.box          binary
*.exe          binary
*.dll          binary
*.so           binary
*.a            binary
#-----
.gitattributes export-ignore
.gitignore     export-ignore

```

GitHub reference: Site-level gitignore

Site-level .gitignore:

```

#Generated by hcigitinit, 2020-09-20-14-05-22
-----

Cloverleaf site-level .gitignore
Runtime dependencies ignore list
#.gitignore backups
/.gitignore.*
#Executable dependency
/exec/databases/
/exec/errors/
/exec/hcilockmgr/
/exec/hcimonitord/
/exec/hcimsgarchive/
/exec/processes/
/exec/mid.ctr
/exec/monitorShmemFile
/exec/sem_*
#Formats
/formats/generatesql.log
/formats/generatesql.err
#Locks
/lock/
#Revisions
/revisions/

```

GitHub reference: Root-level gitignore

Root-level .gitignore:

```
#Generated by hcigitinit, 2020-09-20-14-05-22
#Cloverleaf root-level .gitignore
#The content is encouraged to be constantly refreshed to
#reflect the needs from Cloverleaf application development
#-----

#exclude everything in root
/*

#except
!.gitignore
!.gitattributes
!rootInfo

# and except
!/Alerts
!/AppDefaults
#!/archiving
!/auto-start
!/bitmaps

!/bin
/bin/*
!/bin/allowlist.db
!/bin/sys_cmds.dat

#!/box
!/client
/client/*
!/client/brand
!/client/certs
!/client/ScriptEditor
!/client/*.ini
!/client/*.xml

!/clgui
/clgui/*
!/clgui/images

#!/contrib
#!/debuginfo

!/dicom
/dicom/*
!/dicom/dicomcfg.xml

!/eoalias
#!/formats
!/java_uccs

!/lib
/lib/*
!/lib/launchpad

#!/pdl
!/python
!/productinfo
!/scripts

!/security
/security/*
!/security/certs
!/security/data
!/security/java.*
!/security/*.xml
!/security/*.ini
```

```
!/server
/server/*
!/server/audit
!/server/certs
!/server/template
!/server/*.ini
!/server/java.*
!/server/*.xml
!/server/user_cmds.dat
!/server/cipher.txt
!/server/tomcat
/server/tomcat/*
!/server/tomcat/webapps
!/server/tomcat/conf

#!/siteProto
#!/Tables
!/tcl
/tcl/*
!/tcl/lib
/tcl/lib/*
!/tcl/lib/localInit.tcl

!/tclprocs
!/templates
!/version
!/usercmds
!/xslt

#except site, helloworld
!/helloworld
```

Server administration

Note: A security server installation adds the **Host Server** and **Security Server** tabs to the Server Administration GUI. The **Host Server** tab contains sub-tabs which are also present in no security and basic security installations. The **Security Server** tab contains sub-tabs available only in a security server installation (advanced security).

The system tools aid administrators in configuration. Some of these actions affect an entire site. Others affect only clients running from a specific machine.

Separate `.ini` files are maintained for users and server administrators. Users can configure GUI aesthetics and options and administrators can configure server parameters. Administrators also know when to run select administrative GUIs based on the security mode.

System administrators have different parameters to configure than users. Administrators can access several different GUIs and sometimes different `.ini` files. The Server Administration GUI is used to configure server `.ini` options and invoke select administration GUIs.

The Server Administration GUI has a different main panel based on the install. This GUI is used to configure the settings of the host server (`/integrator/server.ini`), or the settings of the Security Server (`/integrator/security.ini`).

- If more than one server is configured, then the main panel shows tabs to differentiate which server is currently being configured.
- If only one server is configured, then no tabs exist.

Points to remember:

- The title bar shows an asterisk (*) at the end of the GUI name to indicate a requirement to save.
- Changes to the environments on the **General** tab are refreshed immediately. All other settings require a host server reboot.
- When an attempt is made to exit the Server Administration GUI and there is unsaved work, a confirmation dialog box opens. This is a warning that an exit is attempted with unsaved work. You can exit or return to the GUI.

Security server settings are:

- Firewall settings. These are on the **Firewall** tab.
- JVM arguments. These are on the **Advanced** tab.

For security server only installs, access to Server Administration does not require a log-in. The host server is not present to perform the log-in. Access permissions are restricted by operating system protection. If a user without privileges has access to the security server machine, then security has already been compromised. The Server Administration GUI, unlike other administration GUIs, can be run on a security server only install.

Setting the environment

When a user opens the IDE or a stand-alone GUI, the IDE/stand-alone GUI is automatically opened to the last site visited. If this is the first invocation for a user, then the **Select a Server and Environment** dialog box opens.

You can select another site from which to work on the **Select a Server and Environment** dialog box. This is opened by selecting **Server > Change** on the IDE.

The IDE or stand-alone GUIs can be run in local or remote mode. Running the IDE or stand-alone GUIs in the local mode, on the same computer as the host server, is better for performance. This includes when the security server is running.

For command-line users, use `setsite` to select the site with which to work.

Working with sites

Add or delete new sites using the **Server Administration** tool. Sites are added under the system root.

Note: Notify production staff before deleting sites.

You can immediately access a new site without shutting down and restarting by selecting the **Server > Change** menu bar option. Then, open the **Select a Server and Environment** dialog box.

Command-line users

Use `hcisiteinit` to add site folders in the installation.

Sites are added to the installation under the system root.

Note: Do not edit `server.ini` unless directed to do so by Support.

Renaming a site

To rename a site:

- 1 Stop all daemons and processes in the site.
- 2 Rename the site directory.
- 3 Update the `server.ini` file.
- 4 Update all encryption keys for the error database, recovery database, and SMAT database files.
- 5 Update the `siteSecurityInfo` file.

Security

This GUI runs only in local mode. In secure modes, when a host server is present, you must log in as part of the existing security structure in the system. For advanced security, run permission is required to invoke the GUI. Permissions are made in the **ACL** (Access Control List) tab of the ACL/Role Manager.

The periodic refresh of the host server environment is restricted to changes in environment settings only. Other changes to `server.ini` when the host server is running are ignored. This is a safety precaution to prevent any security holes that might be created by on-the-fly changes of security settings within `server.ini`.

Web-based server administration

The **Server Administration > Run** menu has options for:

- ACL Role Mgr
- Audit Log GUI
- Certificate Manager GUI
- Cloverleaf Wizard
- Site Init GUI
- Upgrade Utility
- Update Password for Host Server Certificate
- Update Password for Security Server Certificate

These options are outside of the open **Server Administration** tool. Instead, they directly open the web-based user interface.

Advanced security host server settings

The host server Environments settings are held in the host server configuration file (`server.ini`) located in the installation path `\integrator\server`. The format for the portion of the file to edit is:

```
[general]
[exports]
environs=e:\cis6.2\integrator\site1;e:\cis6.1\integrator\site2
[logging]
Cloverleaf_server_category=info
Cloverleaf_server_level=brief
```

The administrator edits the `environs` entry under the `[exports]` property tag. Multiple sites are concatenated to the `environs=` key using `;`.

Use a text editor that does not introduce editor control characters to the file (for example, Notepad or vi).

Clients can immediately access, without shutting down and restarting, the new site by selecting **Server > Change**.

Updating certificate passwords

If the host server or security server certificate password has changed, then you must update the password in `server.ini`/`security.ini`. After the update, the host server or security server can successfully start.

If the password is not updated, then there is a mis-match and an error results due to the invalid password in the `.ini` file. If there is a mis-match in passwords, then the host server/security server do not start.

In pre-5.8 versions, you could directly change the password value that was saved in `server.ini` or `security.ini`. This is because the password value was formatted in plain text. In version 5.8 and later, the password value in `server.ini` and `security.ini` is encrypted.

Therefore, the **Enter Certificate Password** dialog box is used to change the password value for the host server or security server certificate `.ini` files.

Host server administration

On the machine where the host server is installed, if the host server is under basic or advanced security, then **Server Administration** checks whether the password is valid for the host server certificate in `server.ini`.

If not, then a dialog box opens to prompt you to input the correct password.

- An error message opens if you specify an incorrect password.
- You can make three attempts to specify the correct password. If you do not specify the correct password by third attempt, then a message dialog box opens and the application exits.

After the valid password is specified, the **Server Administration** tool saves the password in `server.ini`.

If you do not know the correct password, then you can click **Skip**, or exit by clicking **Cancel**.

Run menu password options

On the **Run** menu are one or both of these options:

- **Update Password for host server Certificate**
This is only when running basic/advanced security.
- **Update Password for security server Certificate**
This is only in security server installations.

Host server certificate password update

Selecting **Run > Update Password for host server Certificate** opens the **Update Certificate Password** dialog box, where you can update the password value for the host server certificate in `server.ini`.

Security server certificate password update

Selecting **Run > Update Password for security server Certificate** opens the **Update Certificate Password** dialog box, where you can update the password value for the security server certificate in `security.ini`.

On machines where the security server is installed, the server administrator checks whether the password value for the security server certificate in `security.ini` is valid.

Host server and security server certificates password update

For machines that have both the host server and security server installed, the **Server Administration** tool performs these tasks when it starts:

- When the host server is running under basic or advanced security, the **Server Administration** tool checks the validity of the host server certificate password value in `server.ini`.
- The Server Administrator also checks whether the password value for the security server certificate in `security.ini` is valid.

After a password value is detected as invalid, the corresponding dialog box prompts for specifying the valid password.

If both password values are invalid, then the **Enter Certificate Password** dialog box opens with fields for both the host server and security server certificate passwords.

When an invalid password is specified, an `Invalid password for <Host/Security> Server certificate` message is given.

- After specifying the correct password and clicking **OK**, the **Security Server Certificate Password** field becomes uneditable, because that password value is correct and already saved.
- If you do not know the correct password, then specify the host server certificate password or skip by clicking **Skip**.

If both passwords are invalid, then an `Invalid password for host server and security server certificates` is given.

Email setting dialog box

Selecting **Options > Mail** opens the **Email Setting** dialog box. This is where you set the SMTP server and email account settings for the host server email address.

These settings are saved in `email.xml` under `$HCIR00T/server`.

Email template files are located in `$HCIR00T/server/template`.

Host Server General tab

Use this tab to update the environment in which to work. The environments are viewed in the **Server > Change** menu of the IDE. Unlike most other options, this option takes effect immediately.

To create a site, use the **Site Init** GUI. Then, specify the path to the new site name in the text box and click **Add**. The new entry displays on the list.

Points to remember:

- **Add** is enabled only when there is text in the accompanying text field.
- **Remove** is enabled only when an environment is selected in Environments.
- When no or basic security are run without an audit server, the IDE can be opened in local mode without a running host server.
- In advanced security, the IDE cannot run in local mode without a running host server.

Note: A validation failure results in an error message and canceling the **Save** operation.

Host Server Firewall tab

Use this tab to update the TCP/IP and net address translation port settings.

The Java Remote Method Invocation Application Programming Interface (API), or Java RMI, is a Java API that performs the object-oriented equivalent of remote procedure calls (RPC). It does this with support for direct transfer of serialized Java objects and distributed garbage collection.

The original implementation depends on Java Virtual Machine (JVM) class representation mechanisms and thus it only supports making calls from one JVM to another. The protocol underlying this Java-only implementation is known as Java Remote Method Protocol (JRMP).

To support code running in a non-JVM context, a CORBA version was later developed. Usage of the term RMI may denote solely the programming interface. It could also signify both the API and JRMP. The term RMI-IIOP, read as RMI over IIOP, denotes the RMI interface delegating most of the functionality to the supporting CORBA implementation.

TCP/IP settings

TCP/IP settings include:

- **RMI Registry Port** specifies one of two TCP/IP ports that are required for the system components to communicate with one another. The other port is the **RMI Object Port**. The default port is 13021.
If the default is changed, then the system clients must have their `client.ini` file manually modified to match this setting.
If **RMI Registry Port** is cleared, then the current set value is retrieved and populated in the text box before the save.

If the default RMI value is set, then the `server.ini [firewall] rmi_registry_port` parameter is removed from `server.ini`.

- **RMI Object Port** specifies the second of two TCP/IP ports that are required for the system components to communicate with one another.

The default is for the host server to randomly create sockets on any available TCP/IP port. After this option is set, the host server uses this TCP/IP port for all new socket connections. If no port is specified, then `Not Set` populates the text box.

If **RMI Object Port** is cleared, then the text `Not Set` populates the text box.

The `server.ini [firewall] host_server_default_port` parameter is removed from `server.ini`.

- **RMI Callback Port**

To use this setting, you must open the RMI callback port on the firewall and set this RMI callback port on the host server side. This port is transparent to the end-user. That is, the end-user has no requirement to do any additional setting on the client side.

Under non-security mode, you can set the RMI callback port to be the same as the RMI object port. You can also reuse the RMI object port for RMI callback service. If you set a valid RMI callback port, then the host server starts the RMI callback service. This service can pass through the firewall to support the RMI callback behavior. Otherwise, the RMI callback service is stopped.

In security mode, you cannot reuse the RMI object port for RMI callback service. The RMI object port and RMI callback port must be different.

If no port is specified, then `Not Set` populates the text box.

Note: This setting is only required when there is a firewall on the client side.

Net address translation port

Use **Exported Server Address** when a host server is behind a firewall and clients cannot access the IP address directly.

If **Exported Server Address** is cleared, then the text `Not Set` populates the text box. The `server.ini [firewall] rmi_exported_server_port` parameter is removed from `server.ini`.

For security purposes, the host server exports its IP address to any clients that communicate with it. This setting overrides the value exported.

When GUI access is not available

For users who do not have GUI access, sample lines for inclusion can be added to the `[firewall]` section of `server.ini`. After adding these lines to `server.ini` lines, bounce the host server to have it take effect.

- If the `[firewall]` section already exists, then make the additional entries there.
- If the `[firewall]` section does not exist, then it must be added. This can be added to the end of `server.ini`.

The `[firewall]` lines and their GUI counterparts are:

- `host_server_default_port`

- **RMI Object Port**
- **Firewall** tab
- tunnel_port
 - **RMI Callback Port**
 - **Firewall** tab
- rmi_exported_server_port
 - **Exported Server Address**
 - **Firewall** tab

This is the address of the server and not a port number.

- monitord_server_use
 - **Host Server routes traffic**
 - **Advanced** tab

Override Default Settings must be selected before this can be selected.

- monitord_server_buffer_size
 - **MonitorD Message Buffer Size**
 - **Advanced** tab

Host Server Advanced tab

MonitorD traffic options include:

- **Use Default Settings**

System clients can receive MonitorD traffic directly or through the host server. In the [firewall] section of `server.ini`, `monitord_server_use` is not defined.

 - When **Use Default Settings** is selected, **Host Server Routes Traffic** and **MonitorD Message Buffer Size** are disabled.
 - When **Use Default Settings** is selected with no security, clients receive MonitorD traffic directly.
 - When **Use Default Settings** is selected with basic or advanced security, clients receive MonitorD traffic through the host server. That is, the host server routes MonitorD traffic.
- **Override Default Settings**

How MonitorD traffic is routed depends on if **How Server Routes Traffic** is selected.

In the [firewall] section of `server.ini`, `monitord_server_use=true/false` with `monitord_server_buffer_size=xxxx`, where `xxxx` is an integer. The default is 2000.
- **Host Server Routes Traffic**

This option is disabled by default. Select this to receive traffic through the host server. The advantages of receiving traffic through the host server are:

 - A host server can be configured to use two specified TCP/IP ports.
 - Host server traffic is secure for basic and advanced security.

As clients can receive MonitorD traffic directly or through the host server:

- Clients receive MonitorD traffic directory if **Host Server Routes Traffic** is cleared, even if the host server is running with basic or advanced security.

The `server.ini` settings are:

```
[firewall]
monitord_server_buffer_size=2000
monitord_server_use=false
```

On the GUI, **Host Server Routes Traffic** and **MonitorD Message Buffer Size** are set to their default values.

- Clients receive MonitorD traffic through the host server, where the host server routes MonitorD traffic, if **Host Server Routes Traffic** is selected. This also applies if the host server is running with no security.

The `server.ini` settings are:

```
[firewall]
monitord_server_buffer_size=2000
monitord_server_use=true
```

- **MonitorD Message Buffer Size**

This indicates the maximum amount of MonitorD messages that a host server can store in its internal buffers. If this maximum is exceeded, then the host server assumes the client connection has been lost.

This option is available only when **Override Default Settings** is selected. The default is 2000.

For the remainder of options, see:

- [Open Tools options](#)
- [XLT debug traffic](#)
- [JVM options](#)
- [Log-out options](#)

Open Tools options

At times, operators open a tool to view a configuration, but unintentionally lock the file.

Select **Open Tools as read-only** to set the default for tools to open as read-only. This check box is cleared by default.

This setting is stored in the `[general]` section of `server.ini`.

This option is also available from the Site Manager context menu when a tool node is right-clicked.

If a tool is already open, then this option is disabled; otherwise, it is enabled. When there are multiple selections of Site Manager tree nodes, this option does not display.

XLT debug traffic

This configures the XLT debug connection.

When **Host Server routes traffic** is selected, the key in `server.ini` is `true`. When it is cleared, the key is `false` and this is a direct connection.

```
[firewall]
xltdebug_server_use=true
```

JVM options

Arguments can be passed to the JVM during invocation. The arguments are specific to a JVM. These options vary by operating system; specify `java -h` on the command line to see the options.

Note: If other items are included on the **JVM Arguments** line, then they must be separated by spaces.

Examples of other JVM arguments include:

- `-Xmixed`
Mixed mode running (default).
- `-Xint`
Interpreted mode running only.
- `-Xbootclasspath:` Directories and zip/jar files that are separated by ;
Set search path for bootstrap classes and resources.
- `-Xbootclasspath/a:` Directories and zip/jar files that are separated by ;
Append to end of bootstrap class path.
- `-Xbootclasspath/p:` Directories and zip/jar files that are separated by ;
Prepend in front of bootstrap class path.
- `-Xnoclassgc`
Disable class garbage collection.
- `-Xincgc`
Enable incremental garbage collection.
- `-Xloggc: file`
Log GC status to a file with time stamps.
- `-Xms size`
Set initial Java heap size.
- `-Xmx size`
Set maximum Java heap size.
- `-Xss size`
Set java thread stack size.
- `-Xcheck:jni`

Perform additional checks for JNI functions.

Log-out options

Select **Automatically log IDE off after idling for n Minute(s)** to automatically log out the user after the configured time period. The time-out is measured from the last activity.

When this is selected, the time field is also enabled. The default value is 30 (minutes). You can only use positive integers. Settings are stored in `server.ini`.

This function is enabled/disabled by the administrator. By default, this is cleared.

If any configurations are not saved and this option is selected, then unsaved configurations are saved by "emergency save." You can restore them when the IDE is reopened.

For example:

- 1 The administrator enables **Automatically log IDE off after idling for n Minutes** using 30 minutes.
- 2 UserA opens the IDE and makes updates in `NetConfig` or other configuration files.
- 3 UserA has an emergency and leaves the IDE inactive more than 30 minutes.
- 4 Cloverleaf then automatically logs out userA's IDE. To keep userA's modifications, emergency save is automatically used when the IDE is closed.

Heap size

The default maximum heap size is half of the physical memory up to a physical memory size of 192 MB. Otherwise, one-fourth of the physical memory up to a physical memory size of 1 GB.

For example, if your machine has 128 MB of physical memory, then the maximum heap size is 64 MB. A greater than or equal to 1 GB of physical memory results in a maximum heap size of 256 MB.

If the maximum heap size is not enough for the client, then you can specify an increased maximum heap size. To do this, use `-Xmx` in the **Java Arguments** field in **IDE > Client Preferences > Advanced tab**.

For example, `-Xmx512m`.

Security Server RMI registry port

When the host server is using advanced security, the **Security Server RMI Registry Port** field displays. In this field, you can set the RMI Registry port that is used by the security server. If the host server is under no or basic security, then this field does not display.

By default, the host server uses the host server RMI Registry port number to connect to the security server. If any valid value is specified in this field, then the host server attempts to connect to the security server with that value.

The default value for this field is `Not Set`. Usually, there is no requirement to set this port.

If the host server tries to connect to different versions of security server, then the host server must change its RMI registry port number. This might affect the IDE client connectivity and might not be necessary.

To use a different port number for security server connection purposes, specify the port number in the field.

This setting is especially important when multiple versions of the system are running on the same machine. The host server registry port number must be different for these host servers to work correctly. They must use another port number to connect to a shared security server box. In that case, you also must configure this setting for each host server.

Host Server Monitor tab

These configuration options are available on the Monitor tab:

- **Audit Server**

This enables/disables the audit server. Local clients can run without a host server when the audit server is disabled.

By default, this option is off and must be turned on manually before logging begins.

In advanced security, a host server/GUI always runs as if configured for an audit server. This applies even if the host server was configured to not run an audit server.

When the host server is not configured to run an audit server, the host server lists a warning message in the `server.log` file. This indicates that it has nevertheless started an audit server.

- **Global Monitor Server**

This enables/disables the host server to interact with Global Monitor.

- **CSC Monitor Server**

This enables/disables the host server to interact with Secure Courier. The CSC Monitor server is similar to audit server or Global Monitor server.

This must be enabled to use CSC server monitoring.

In the **CSC RMI Port** field, specify the RMI port on which the CSC server exposes its JMX objects.

If this field is empty, then the CSC RMI default port is used.

The configurations of CSC Monitor server are stored in the `[general]` section of `server.ini`.

Suspended User Cache tab

This tab lists the suspended users whose log-in failed over the maximum number of attempts.

The user's certificate and private key files must be on the Host Server. The Cloverleaf API uses these credential files to connect to the Host Server.

Click **Refresh** to refresh the list of suspended users.

Before the suspension time elapses, Administrators can unsuspend or re-enable the suspension by clicking **Unlock** or **Relock**, respectively.

The maximum number of log-in attempts is five (5).

The log-in suspension maximum time is 15 minutes.

When the number of log-in attempts has been exceeded, a message is given, stating:

You have exceeded the number of log-in attempts. Contact your administrator for assistance.

Host Server Web Server tab

Outside applications can access some internal functions, such as SMAT message resend or error message resend.

These internal functions are exposed as web services, so that the outside applications can access them through Simple Object Access Protocol (SOAP).

Additional options are:

- **Launch web server on Host Server startup**

When this is selected, the web server starts along with the host server. If this cleared, then **Enable web services** is also cleared. The default is selected.

- **Enable Cloverleaf API**

The Cloverleaf API uses a RESTful web service architecture and (JavaScript Object Notation (JSON) as the data interchange format. Through this, you can create, edit, and delete Cloverleaf objects that were traditionally only available in the Cloverleaf GUI.

- **Enable Cloverleaf API Documentation**

Enable Cloverleaf API must be selected to enable this check box.

By default, **Enable Cloverleaf API Documentation** is cleared.

The CLAPI web page includes the CLAPI SDK document and examples. By default, it is disabled. To enable the **CLAPI** web page, the host server must be running in security mode.

When the Cloverleaf host server is started with **Enable Cloverleaf API Documentation** enabled, all Cloverleaf APIs are listed in HTML through <https://hostservername:15047/clapi/>. This provides details about request parameters and responses for each API.

You can also enter API parameters and click **Try it out** in the API window.

The Resources Available page lists available APIs and their location.

For information on CLAPI, see [Cloverleaf API](#).

- **Port**

Port 15040 is the default for the Site Document.

By default, the HTML files that are generated by the Site Document are put under `$HCIR00T/server/tomcat/webapps/sitedoc`.

- Any changes enable **Save**. After you click **Save**, you must restart the host server before the change takes place.
- The HTTP port defaults to 15040, HTTPS defaults to 15043, and the Tomcat server shutdown port defaults to 15045. To change these ports, go to `$HCIR00T/server/tomcat/conf/server.xml`.

Web services respect the host server's security configuration. If the host server is configured as no security, then no security is used for web services. If the host server is configured as basic or advanced security, then Transport Security (SSL) is used for web services. SSL provides a point-to-point security.

Web services and security

Web services respects the system's security configuration.

- If the host server is configured as no security, then no security is used for web services.
- If the host server is configured as basic or advanced security, then Transport Security (SSL) is used for web services. Transport Security (SSL) provides a point-to-point security.

When upgrading security, the Security Upgrade tool updates the security configurations for web services according to the target security mode.

- If a security upgrade changes the security mode to no security, then it removes security-related configuration from web services.
- If a security upgrade changes the security mode to basic security or advanced security, then it adds security configuration to web services. It also imports the host server's public key certificate and private key certificate into a keystore file, which is required for Transport Security (SSL).

See [Common web services](#).

See [SMAT message resend related web services](#).

See [Error message resend related web services](#).

Common web services

The ports can be changed using the **Web Server** tab, or going to `$HCIR00T/server/tomcat/conf/server.xml` and manually changing the port.

- `String[] getSiteList()` retrieves the site list.
This returns a string array that contains all the site names.
- `String[] getSiteListExt(HashMapWrapper paramsMap)` retrieves the site list, where `paramsMap` can be set to null.
This returns a string array that contains all the site names.
- `String[] getProcessList(String siteName)` retrieves the process list from the specified site, where `siteName` is the name of the site.
This returns a string array that contains all the processes under the specified site.

- `String[] getProcessListExt(HashMapWrapper paramsMap)` retrieves the process list from the specified site, where `paramsMap` is the site name. The key is `siteName` and value is the site name.
This returns a string array that contains all the processes under the specified site.
- `String[] getThreadList(String siteName, String processName)` retrieves the thread list from the specified process that is under the specified site, where:
 - `siteName` is the site name.
 - `processName` is the process name.This returns a string array that contains all the threads under the specified site.
- `String[] getThreadListExt(HashMapWrapper paramsMap)` retrieves the thread list from the specified process under the specified site, where `paramsMap` is the site name and process name in a map.
 - `siteName` is the name of the site name.
 - `processName` is the name of the process name.This returns a string array that contains all the threads under the specified site.

SMAT message resend related web services

Through these web services interfaces, the outside applications can retrieve the message ID list based on the search conditions. The applications can then dump the message content and metadata that are based on the message ID to view and decide whether to change them for resend.

```
String[] SMAT_getIndexFileList(String siteName, String  
                             processName)
```

This retrieves the index file name list from the specified process under the specified site. The outside application accesses this interface to retrieve the index file name list and decides to query the message IDs from which index file.

- `siteName` is the name of the site.
- `processName` is the name of the process.
- This returns a string array that contains the index file name list.

```
String[] getSMATIndexFileListExt (HashMapWrapper  
                                paramsMap)
```

This retrieves the index file name list from the specified process under the specified site. The outside application accesses this interface to retrieve the index file name list and decides to query the message IDs from which index file.

- `paramsMap` is a map that contains the site name and process name. Their keys are `siteName` and `processName`.
- This returns a string array that contains the index file name list.

```
String[] SMAT_getCyclingIndexFileList(String siteName, String  
                                     processName)
```

This retrieves the cycling index file name list from the `smatHistory` folder of the specified process under the specified site. The outside application accesses this interface to retrieve the cycling index file name list and decides to query the message ID from which index file.

- `siteName` is the name of the site.
- `processName` is the name of the process.
- This returns a string array containing the cycling index file name.

```
String[] getSMATCyclingIndexFileListExt (HashMapWrapper
    paramsMap)
```

This retrieves the cycling index file name list from the specified process under the specified site. The outside application accesses this interface to retrieve the cycling index file name list and decides to query the message ID from which index file.

- `paramsMap` is a map that contains the site name and process name, whose keys are `siteName` and `processName`.
- This returns a string array containing the cycling index file name.

```
int SMAT_getMessageCount(String siteName, String
    indexFileName)
```

This returns the message count of the specified index file under the specified site.

- `siteName` is the name of the site.
- `indexFileName` is the name of the index file.

```
int getSMATMessageCountExt (HashMapWrapper
    paramsMap)
```

This returns the message count of the specified index file under the specified site.

`paramsMap` is a map that contains the site name and index file name, whose keys are `siteName` and `indexFileName`.

```
String[] SMAT_getMessageIds(String siteName, String
    indexFileName, int offset, int length)
```

This searches the specified number of message IDs beginning from `offset`. It then returns the matched message ID list in a string array.

- `siteName` is the name of the site.
- `indexFileName` is the name of the index file.
- `offset` is the index of the first message ID to search. `offset` represents the physical location in the SMAT index file.
- `length` is the number of the message ID to search.
- This is a high-performance query interface. The precondition is that the outside applications know the position range of the messages in which they are interested.

```
String[] SMAT_getMessageIds(String siteName, String
    indexFileName, SMATMsgSearchCriteria searchCriteria)
```

This queries the message IDs under the specified search criteria in the specified index file of the specified site.

- `siteName` is the name of the site.
- `indexFileName` is the name of the index file where the message IDs are queried.
- `SearchCriteria` is the SMAT message search criteria.
- `SMATMsgSearchCriteria` defines the condition for the source thread name, the message ID range, the sending date range, or the message search string.
- This returns a string array containing the matched message IDs.

```
String[] getSMATMessageIdsExt (HashMapWrapper
    paramsMap)
```

- If `paramsMap` contains `offset` and `length`, then this searches the specified number of message IDs beginning from `offset`. It then returns the matched message ID list in a string array.
- `offset` represents the physical location in the SMAT index file. This is a high-performance query interface. The precondition is that the outside applications know the position range of the messages in which they are interested.
- If `paramsMap` contains `searchCriteria`, then this queries the message IDs under the specified search criteria in the specified index file of the specified site. It then returns a string array containing the matched message IDs.
- `searchCriteria` is `SMATMsgSearchCriteria` type. This defines the condition for the source thread name, message ID range, sending date range, or message search string.
- `paramsMap` is a map that contains the site name, index file name, SMAT message search criteria/index of the first message, and number of the message.
- Their keys are `siteName`, `indexFileName`, `searchCriteria`, `offset`, and `length`.
 - `siteName` and `indexFileName` are required.
 - `offset` and `length` are always paired.

This pair of parameters and `searchCriteria` are optional.

- This returns a string array containing the matched message IDs.

```
String SMAT_getMessageId(String siteName, String
    indexFileName, int index)
```

This searches the message at the specified index in the specified index file and returns the corresponding message ID. This is a high-performance query interface.

- `siteName` is the name of the site.
- `indexFileName` is the name of the index file.
- `index` is the physical location in the SMAT index file, beginning from 0.

```
String getSMATMessageIdExt (HashMapWrapper paramsMap)
```

This searches the message at the specified index in the specified index file and returns the corresponding message ID. This is a high-performance query interface.

`paramsMap` is a map that contains the site name, index file name, and the physical location in the SMAT index file, beginning from 0, whose keys are `siteName`, `indexFileName`, and `index`.

```
Message SMAT_getMessage(String siteName, String
    indexFileName, String mid)
```

This matches the message according to the specified ID in the specified index file. Returns the message object that contains the content and metadata referencing the SOAP attachments. The format of the message content and metadata is exactly the same as the one used by the SMAT message resend command.

- `siteName` is the name of the site.
- `indexFileName` is the name of the index file.
- `mid` is the message ID.

```
Message SMAT_getMessage(String siteName, String
    indexFileName, int index)
```

This searches the message at the index from the specified index file. It then returns a message object that contains the content and metadata referencing the SOAP attachments. The format of the message content and metadata is exactly the same as the one used by the SMAT message resend command.

- `siteName` is the name of the site.
- `indexFileName` is the name of the index file.
- `index` is the message index.

```
Message getSMATMessageExt(HashMapWrapper
    paramsMap)
```

This searches a message using the specified parameters and returns a message object containing the message content and metadata input references.

- If `paramsMap` contains `siteName`, `indexFileName`, and `messageId`, then this matches the message according to the specified ID in the specified index file. It then returns the message object that contains the content and metadata referencing the SOAP attachments. The format of the message content and metadata is exactly the same as the one used by the SMAT message resend command.
- If `paramsMap` contains `siteName`, `indexFileName`, and `index`, then this searches the message at the index from the specified index file. It then returns the message object that contains the content and metadata referencing the SOAP attachments. The format of the message content and metadata is exactly the same as the one used by the SMAT message resend command.
- `paramsMap` is a map that contains `siteName`, `indexFileName`, and `messageId` or `index`, where `messageId` and `index` are optional.

```
void SMAT_resendMessage(String siteName, String
    processName, String threadName, Context context, SMATMsgType type, int
    priority, DataHandler messageContent, ContentFormat contentFormat) throws
    ResendFailedException
```

This resends the messages with the specified content to the specified thread. The format of the message content is exactly the same as the one used by the SMAT message resend command.

- `siteName` is the name of the site.
- `processName` is the name of the process.
- `threadName` is the name of the thread to which the message is resent.
- `context` is the context. This can be `Context.IB_PRE_TPS`, `Context.IB_POST_TPS`, `Context.OB_PRE_TPS`, or `Context.OB_POST_TPS`.
- `type` is the message resend type. This can be `SMATMsgType.DATA` or `SMATMsgType.REPLY`.
- `priority` is the resend priority.

- `messageContent` is a `DataHandler` instance which references an input stream of the message content.
- `contentFormat` is the message content format. This can be `ContentFormat.LEN10`, `ContentFormat.EOF`, or `ContentFormat.EOL`.
- **Exception:** `ResendFailedException` is thrown when the resend fails.

```
void SMAT_resendMessage(String siteName, String
    processName, String threadName, Context context, SMATMsgType type, int
    priority, DataHandler messageContent, ContentFormat contentFormat, DataHandler
    metaData) throws ResendFailedException
```

This resends the messages with the specified content and metadata to the specified thread. The format of the message content and metadata is exactly the same as the one used by the SMAT message resend command.

- `siteName` is the name of the site.
- `processName` is the name of the process.
- `threadName` is the name of the thread name to which the message is resent.
- `context` is the context. This can be `Context.IB_PRE_TPS`, `Context.IB_POST_TPS`, `Context.OB_PRE_TPS`, or `Context.OB_POST_TPS`.
- `type` is the message resend type. This can be `SMATMsgType.DATA` or `SMATMsgType.REPLY`.
- `priority` is the resend priority.
- `messageContent` is a `DataHandler` instance which references an input stream of the message content.
- `contentFormat` is the message content format. This can be `ContentFormat.LEN10`, `ContentFormat.EOF`, or `ContentFormat.EOL`.
- `metaData` is a `DataHandler` instance which references an input stream of the meta data.
- **Exception:** `ResendFailedException` is thrown when the resend fails.

```
void SMAT_resendMessage(String siteName, String
    processName, String threadName, Context context, SMATMsgType type, int
    priority, String indexFilePath, String[] msgIds) throws
    ResendFailedException
```

This resends the messages with the specified message IDs. The resend is processed without any message content and metadata changed.

- `siteName` is the name of the site.
- `processName` is the name of the process.
- `threadName` is the thread name to which the message is resent.
- `context` is the context. This can be `Context.IB_PRE_TPS`, `Context.IB_POST_TPS`, `Context.OB_PRE_TPS`, or `Context.OB_POST_TPS`.
- `type` is the message resend type. This can be `SMATMsgType.DATA` or `SMATMsgType.REPLY`.
- `priority` is the resend priority.
- `indexFilePath` is the path of the index file where the messages are resent.
- `msgIds` is the `String` array containing the IDs of the message that would be resent.
- **Exception:** `ResendFailedException` is thrown when the resend fails.

```
void SMAT_resendMessage(String siteName, String
    processName, String threadName, Context context, SMATMsgType type, int priority,
    DataHandler messageContent, ContentFormat contentFormat, String encoding) throws
    ResendFailedException
```

This resends the message with the specified content to the specified thread.

- `siteName` is the site name.
- `processName` is the process name.
- `threadName` is the thread name.
- `context` is the context. This can be `Context.IB_PRE_TPS`, `Context.IB_POST_TPS`, `Context.OB_PRE_TPS` or `Context.OB_POST_TPS`.
- `type` is the type. This can be `SMATMsgType.DATA` or `SMATMsgType.REPLY`.
- `priority` is the priority.
- `messageContent` is the message content.
- `contentFormat` is the message content format. This can be `ContentFormat.LEN10`, `ContentFormat.EOF` or `ContentFormat.EOL`.
- `encoding` is the encoding of the message data. This can be null.
- **Exception:** `ResendFailedException` is thrown when any exception happens after 5.8.5.0.

```
void SMAT_resendMessage(String siteName, String
    processName, String threadName, Context context, SMATMsgType type,int priority,
    DataHandler messageContent,ContentFormat contentFormat,String encoding,
    DataHandler metaData) throws ResendFailedException
```

This resends the message with the specified content and metadata to the specified thread.

- `siteName` is the site name.
- `processName` is the process name.
- `threadName` is the thread name.
- `context` is the context, could be `Context.IB_PRE_TPS`, `Context.IB_POST_TPS`, `Context.OB_PRE_TPS` or `Context.OB_POST_TPS`.
- `type` is the type, could be `SMATMsgType.DATA` or `SMATMsgType.REPLY`.
- `priority` is the priority.
- `messageContent` is the message content.
- `contentFormat` is the message content format, could be `ContentFormat.LEN10`, `ContentFormat.EOF` or `ContentFormat.EOL`.
- `encoding` is the encoding of the message data, can be null.
- `metaData` is the meta data.
- **Exception:** `ResendFailedException` is thrown when any exception happens after 5.8.5.0.

```
void resendSMATMessageExt(HashMapWrapper paramsMap) throws
    ResendFailedException
```

This resends the message with the specified content and metadata to the specified thread.

- If `paramsMap` contains `messageContent`, `contentFormat`, `metadata`, and `encoding`, then this resends the messages with the specified message content and metadata.
 - The format of `messageContent` and `encoding` is exactly the same as the one used by the SMAT message resend command.
 - `encoding` can be empty.
- If `paramsMap` contains `messageContent`, `contentFormat`, and `encoding`, then this resends the messages with the specified message content.

- The format of `messageContent` and `encoding` is exactly the same as the one used by the SMAT message resend command.
- `encoding` can be empty.
- If `paramsMap` contains does not contain `messageContent` or `contentFormat`, then this resends the messages with the specified message IDs. The resend is processed without any message content and metadata being changed.
- `paramsMap` is a map that contains `siteName`, `processName`, `threadName`, `context`, `type`, `priority`, `messageContent`, `contentFormat`, `metaData`, `encoding`, `indexFileName`, and `messageIds`.
- `indexFileName` and `messageIds` are always paired.
- Exception: `ResendFailedException` is thrown when the resend fails.

Error message resend related web services

The outside applications can query error message IDs that are based on the search condition. These applications can then dump the corresponding message content and metadata to view. They then resend the error message by accessing the error message resend web services.

```
String[] DB_getMessageIds(String siteName,
    DBMsgSearchCriteria searchCriteria)
```

This queries the messages in the error database under the specified site to meet the search criteria defined in `searchCriteria`. It then returns a string array containing the matched message IDs.

- `siteName` is the name of the site.
- `searchCriteria` is the message query search criteria.
- `DBMsgSearchCriteria` defines the condition for the source/destination thread name, owner name, error state, message type, message ID range, or created time range.

```
String[] getDBMessageIdsExt (HashMapWrapper
    paramsMap)
```

This queries the messages in the error database under the specified site to meet the search criteria defined in `searchCriteria`. It then returns a string array containing the matched message IDs.

- `searchCriteria` is `DBMsgSearchCriteria` type. This defines the condition for the source/destination thread name, owner name, error state, message type, message ID range, or created time range.
- `paramsMap` is a map that contains parameters, whose keys are `siteName` and `searchCriteria`.

```
DataHandler DB_getMessageContent (String siteName,
    DBMsgSearchCriteria searchCriteria, ContentFormat contentFormat)
```

This queries the error database to match the search criteria under the specified site and returns the message content with the specified format.

- `siteName` is the site name under which the error database is queried.
- `searchCriteria` is the database error message search criteria.

- `contentFormat` is the format of the returned message content. This can be one of `ContentFormat.LEN10`, `ContentFormat.EOF`, or `ContentFormat.EOL`.

```
DataHandler getDBMessageContentExt(HashMapWrapper
    paramsMap)
```

This queries the error database to match the search criteria under the specified site. It then returns the message content with the specified format.

- `paramsMap` is a map that contains the site name, search criteria, and format of the returned message content, whose keys are `siteName`, `searchCriteria`, and `format`.
- `format` is one of `ContentFormat.LEN10`, `ContentFormat.EOF`, or `ContentFormat.EOL`.

```
DataHandler DB_getMetaData (String siteName,
    DBMsgSearchCriteria searchCriteria, ContentFormat contentFormat)
```

This searches the error database to match the search criteria and returns the matched metadata.

- `siteName` is the site name under which the error database is queried.
- `searchCriteria` is the database search criteria.
- This returns a `DataHandler` instance referencing the input stream of the matched metadata.

```
DataHandler getDBMetaDataExt(HashMapWrapper
    paramsMap)
```

This searches the Error Database to match the search criteria and returns the matched metadata.

- `paramsMap` is a map that contains site name and search criteria. The keys are `siteName` and `searchCriteria`.
- This returns the `DataHandler` instance referencing the input stream of the matched metadata.

```
Message DB_getMessage (String siteName, DBMsgSearchCriteria
    searchCriteria, ContentFormat contentFormat)
```

This searches the error database under the search criteria and returns the matched message content and metadata. The format of the message content and metadata is the same as the one used in the error message dump command.

- `siteName` is the site name under which the error database is queried.
- `searchCriteria` is the database search criteria.
- `contentFormat` is the format of the returned message content. This can be one of `ContentFormat.LEN10`, `ContentFormat.EOF`, or `ContentFormat.EOL`.

```
Message getDBMessageExt(HashMapWrapper
    paramsMap)
```

This searches the error database under the search criteria and returns the matched message content and metadata. The format of message content and metadata is the same as the one used in the error message dump command.

- `paramsMap` is a map that contains the site name, search criteria, and format of the returned message content, whose keys are `siteName`, `searchCriteria`, and `format`.

- `format` is one of `ContentFormat.LEN10`, `ContentFormat.EOF`, or `ContentFormat.EOL`.

```
void DB_resendMessage(String siteName, String processName,
    MID messageId) throws ResendFailedException
```

This resends the message with the specified message ID.

- The resend command is:

```
hclcmd -p process_name -c ". resend_errordb
mid"
```

- `siteName` is the name of the site.
- `processName` is the name of the process.
- `messageId` is the message ID.
- **Exception:** `ResendFailedException` is thrown when the resend fails.

```
void DB_resendMessage(String siteName, String processName,
    DataHandler metaData) throws ResendFailedException
```

This resends the message from the error database with the modified metadata from the SOAP attachment.

- The command is:

```
hclcmd -p process_name -c ". resend_errordb -m
metadata"
```

- The format of the message content and metadata is the same as the one used in the error message resend command.
- `siteName` is the name of the site.
- `processName` is the process to which the message is sent.
- `metaData` is a `DataHandler` instance which references an input stream of the message metadata.
- **Exception:** `ResendFailedException` is thrown when the resend fails.

```
void DB_resendMessage(String siteName, String processName,
    MID messageId, DataHandler messageContent, ContentFormat contentFormat) throws
ResendFailedException
```

This resends the message content with the specified format. It also resends the metadata of the message that is identified by the specified ID to the specified process.

- The resend command is:

```
hclcmd -p process_name -c ".resend_errordb <mid>
msgContent len10|nl|eof"
```

- `siteName` is the name of the site.
- `processName` is the name of the process.
- `messageId` is the message ID.
- `messageContent` is the message content.

- `contentFormat` is the content format. This can be one of `ContentFormat.LEN10`, `ContentFormat.EOF`, or `ContentFormat.EOL`.
- **Exception:** `ResendFailedException` is thrown when the resend fails.

```
void DB_resendMessage(String siteName, String processName,
    DataHandler metadata, DataHandler messageContent, ContentFormat contentFormat)
    throws ResendFailedException
```

This resends the message with the specified message content and metadata.

- The command is:

```
hcicmd -p process_name -c ". resend_errordb -m metadata
    msgcontent len10|nl|eof"
```

- `siteName` is the name of the site.
- `processName` is the name of the process.
- `metaData` is the metadata.
- `messageContent` is the message content.
- `contentFormat` is the content format.
- **Exception:** `ResendFailedException` is thrown when the resend fails.

```
void DB_resendMessage(String siteName, String processName,
    MID messageId, DataHandler messageContent, ContentFormat contentFormat, String
    encoding) throws ResendFailedException
```

This resends the message content in the specified format into the specified process.

- The resend command is:

```
hcicmd -p <process_name> -c ". resend_errordb mid
    msgContent len10|nl|eof [-e msg_encoding]"
```

- `siteName` is the site name.
- `processName` is the process name.
- `messageId` is the message ID.
- `messageContent` is the message content.
- `contentFormat` is the content format. This can be `ContentFormat.LEN10`, `ContentFormat.EOF` or `ContentFormat.EOL`.
- `encoding` is the encoding of the message data. This can be null.
- **Exception:** `ResendFailedException` is thrown when any exception happens after 5.8.5.0.

```
void DB_resendMessage(String siteName, String processName,
    DataHandler metaData, DataHandler messageContent, ContentFormat contentFormat,
    String encoding) throws ResendFailedException
```

This resends the message with the specified message content and metadata.

- The resend command is:

```
hcicmd -p process_name -c ". resend_errordb -m
metadata msgcontent len10|nl|eof [-e
msg_encoding]"
```

- `siteName` is the site name.
- `processName` is the process name.
- `metaData` is the metadata.
- `messageContent` is the message content.
- `contentFormat` is the content format.
- `encoding` is the encoding of the message data.
- **Exception:** `ResendFailedException` is thrown when the resend fails.

```
void resendDbMessageExt(HashMapWrapper paramsMap) throws
ResendFailedException
```

This resends the message from the error database.

- If `messageContent` is not null, then resend specified message content.
- If `messageId` is not null, then this resends the message content by the specified ID into the specified process.
- If `contentFormat` is not null, then this resends the message content in the specified format into the specified process.
- If `metaData` is not null, then this resends the message with the specified message content and metadata.
- `paramsMap` is a map that contains the site name, process name, message ID metadata, message content, encoding, and content format. The keys are `siteName`, `processName`, `messageId`, `metaData`, `messageContent`, `encoding`, and `contentFormat`.
- **Exception:** `ResendFailedException` is thrown when the resend fails.

Local binding support for CAA-WS

You can use binding for normal network connections.

In some localities there are multiple governmental web services, where in one of the security layers the web server of the government checks from which IP the user is coming.

This feature provides the option to bind to a certain network interface.

When there is no option to configure a binding for NIC to use within a Cloverleaf HA environment, the connection comes from a NODE IP. This is an unacceptable scenario.

There is also an option within CAA-WS to bind to a certain IP. This is similar to the Virtual Cluster IP with all of the normal network connections within Cloverleaf.

"route" command

On AIX, the server administrator can use the `route` command to set up a static route record for binding.

For example, a user has an adapter "A" with an IP address of `a.a.a.a` and an adapter B with an IP address of `b.b.b.b`. The target service is served on IP `c.c.c.c`. These can specify the traffic through the adapter "A". For example::

```
route add c.c.c.c a.a.a.a
```

For details on using the `route` command, see [Using the route command](#) on page 1112.

Using the route command

The `route` command is used to manually manipulate the routing tables.

Usage:

```
route [ -f ] [ -n ] [ -q ] [ -C ] [ -v ] Command [ Family ] [ [ -net | -host ] Destination  
[ -prefixlen n ] [ -netmask [ Address ] ] Gateway ] [ Arguments ] [ -i ] [ -@ WparName ]
```

You can make manual entries into network routing tables with the `route` command. This command distinguishes between "routes to hosts" and "routes to networks" by interpreting the network address of the `Destination` variable. This variable can be specified with a symbolic name or numeric address.

The `route` command resolves all symbolic names into addresses, using the `/etc/hosts` file or the network name server.

Routes to a particular host are distinguished from those to a network by interpreting the internet address that is associated with the destination. The optional `ph -net` and `-host` force the destination to be interpreted as a network or a host, respectively. If the destination has a local address part of `INADDR_ANY` or if the destination is the symbolic name of a network, then the route is assumed to be to a network; otherwise, it is presumed to be a route to a host.

For example:

- `128.32` is interpreted as `-host 128.0.0.32`
- `128.32.130` is interpreted as `-host 128.32.0.130`
- `-net 128.32` is interpreted as `128.32.0.0`
- `-net 128.32.130` is interpreted as `128.32.130.0`

If the route is by way of an interface instead of through a gateway, then the `-interface` argument should be specified. The specified gateway is the address of the host on the common network. This indicates the interface to be used for transmission.

The `-netmask` argument must be followed by an address parameter to be interpreted as a network mask. You can override the implicit network mask generated in the `-inet` case by ensuring this option follows the `Destination` parameter.

All symbolic names that are specified for a destination or gateway are looked up first as a host name, using the `gethostbyname` subroutine. If this fails, then `getnetbyname` is used to interpret the name as a network name.

Note: `route` uses a routing socket and the message types `RTM_ADD`, `RTM_DELETE`, and `RTM_CHANGE`. Only the root user can modify the routing tables.

If the `flush` or `-f` command is specified, then `route` "flushes", or clears the routing tables of all gateway entries. You can choose to flush only those routes whose destinations are of a given address family by specifying an optional `ph` describing which address family.

The `netstat -r` command displays the current routing information contained in the routing tables.

Flags

The route default is a host (a single computer on the network). When the `-net` or `-host` parameter is not specified, but the network portion of the address is specified, the route is assumed to be to a network. The host portion of the address is "0" (zero).

Item	Description
<code>-f</code>	Purges all entries in the routing table that are not associated with network interfaces.
<code>-i</code>	<p>Enables workload-partition-specific routing for the workload partition (WPAR). By default, outgoing network traffic from a WPAR is routed as if it were being sent from the global environment:</p> <ul style="list-style-type: none"> Traffic between addresses that are hosted on the same global system is sent through the loopback interface. Routing table entries that are configured in the global system, including the default route, are used to transmit workload partition traffic. <p>If you enable WPAR-specific routing by specifying the <code>-i</code> flag, then the WPAR creates and uses its own routing table for the outgoing traffic. Routing entries are created automatically for each of the network addresses of the WPAR to accommodate broadcast, loopback, and subnet routes.</p>
<code>-n</code>	Displays host and network names numerically, instead of symbolically, when reporting results of a flush or of any action in verbose mode.
<code>-q</code>	Specifies quiet mode and suppresses all output.
<code>-C</code>	Specifies preference for <code>ioctl</code> calls over routing messages for adding and removing routes.
<code>-v</code>	Specifies verbose mode and prints additional details.
<code>-net</code>	Indicates that the <code>Destination</code> parameter should be interpreted as a network.

Item	Description
-netmask	Specifies the network mask to the destination address. Ensure this option follows the <code>Destination</code> parameter.
-host	Indicates that the <code>Destination</code> parameter should be interpreted as a host.
-prefixlen <i>n</i>	Specifies the length of a destination prefix. This is the number of bits in the netmask.
-@WparName	Displays the network statistics that are associated with the WPAR, that is, @WparName flag. If the @WparName flag is not specified, then the network statistics for all the WPARs are displayed.

Parameters

These arguments are available:

- Arguments
- Commands
- Family
- Destination
- Gateway

Arguments

You can specify one or more of the these arguments, where *n* is specified as a variable to an argument. The value of the *n* variable is a positive integer.

Item	Description
-active_dgd	Enables Active Dead Gateway Detection on the route.
-cloning	Clones a new route.
-genmask	Extracts the length of TSEL, which is used for the generation of cloned routes.
-interface	Manipulates interface routing entries.
-rtt <i>n</i>	Specifies round-trip time.
-rttvar <i>n</i>	Specifies round-trip time variance.
-sendpipe <i>n</i>	Specifies send-window size.
-recvpipe <i>n</i>	Specifies receive-window size.

Item	Description
<code>-allowgroup gid</code>	Specifies a group ID that has permission to use the route. The group ID is added to a list of permitted groups or deleted from a list of denied groups.
<code>-denygroup gid</code>	Specifies a group ID that does not have permission to use the route. The group ID is added to a list of denied groups or deleted from a list of permitted groups.
<code>-stopsearch</code>	Stops searching if a routing table lookup matches the route, but is not permitted to use the route due to group routing restrictions.
<code>-mtu n</code>	Specifies the maximum transmission unit for this route. This overrides the interface <code>mtu</code> for TCP applications as long as it does not exceed maximum <code>mtu</code> for the interface. This flag has no affect on <code>mtu</code> for applications using UDP.
<code>-hopcount n</code>	Specifies maximum number of gateways in the route.
<code>-policy n</code>	<p>Specifies the policy to be used for Multipath Routing. <i>n</i> is number between 1 and 5.</p> <ol style="list-style-type: none"> 1. Weighted Round-Robin 2. Random 3. Weighted Random 4. Lowest Utilization 5. Hash-based <p>If the policy is not explicitly set and multi-path routing is used, then the global <code>no</code> command option called <code>mpr_policy</code> determines the policy that is used. The default policy is Weighted Round Robin. This behaves similar to Round-Robin when the weights are all 1. Although the Default policy is Weighted Round-Robin, when the policy is not set, the network option <code>mpr_policy</code> takes precedence.</p> <p>However, if the policy is explicitly set to <code>WRR</code> then this setting overrides the <code>mpr_policy</code> setting.</p>
<code>-weight n</code>	Specifies the weight of the route that is used for the Weighted policies with the Multipath Routing feature.
<code>-expire n</code>	Specifies expiration metrics used by the routing protocol.
<code>-sssthresh n</code>	Specifies the outbound gateway buffer limit.

Item	Description
<code>-lock</code>	Specifies a meta-modifier that can individually lock a metric modifier. The <code>-lock</code> meta-modifier must precede each modifier to be locked.
<code>-lockrest</code>	Specifies a meta-modifier that can lock all subsequent metrics.
<code>-if ifname</code>	Specifies the interface (<code>en0</code> , <code>tr0</code> ...) to associate with this route so that packets are sent using this interface when this route is chosen.
<code>-xresolve</code>	Emits a message on use for external lookup.
<code>-iface</code>	Specifies that the destination is directly reachable.
<code>-static</code>	Specifies the manually added route.
<code>-nostatic</code>	Specifies the pretend route that is added by the kernel or daemon.
<code>-reject</code>	Emits an ICMP unreachable when matched.
<code>-blackhole</code>	Silently discards packets during updates.
<code>-proto1</code>	Sets protocol specific routing flag number 1.
<code>-proto2</code>	Sets protocol specific routing flag number 2.

Commands

Commands include:

Item	Description
<code>add</code>	Adds a route.
<code>flush</code> OR <code>-f</code>	Removes all routes.
<code>delete</code>	Deletes a specific route.
<code>change</code>	Changes aspects of a route, such as its gateway.
<code>monitor</code>	Reports any changes to the routing information base, routing lookup misses, or suspected network partitionings.
<code>get</code>	Looks up and displays the route for a destination.
<code>set</code>	Sets the policy and weight attributes of a route.

Family

This specifies the address family.

Item	Description
Family	Family specifies the address family. The <code>-inet</code> address family is the default. The <code>-inet6</code> family specifies that all subsequent addresses are in the <code>inet6</code> family.

Destination

This identifies the host or network endpoint for the route.

Item	Description
Destination	Identifies the host or network to which you are directing the route. The <code>Destination</code> parameter can be specified by symbolic name or numeric address.

Gateway

This identifies the gateway for packets.

Item	Description
Gateway	Identifies the gateway to which packets are addressed. The <code>Gateway</code> parameter can be specified by a symbolic name or numeric address.

Security

For RBAC (Role-Based Access Control) users and Trusted AIX users, this command can perform privileged operations. Only privileged users can run privileged operations.

Example

Examples in using `route`:

- Example 1:
To establish a route so that a computer on one network can send a message to a computer on a different network, specify:

```
route add 192.100.201.7 192.100.13.7
```

 - The `192.100.201.7` address is that of the receiving computer (the `Destination` parameter).
 - The `192.100.13.7` address is that of the routing computer (the `Gateway` parameter).
- Example 2:

To establish a route so that you can send a message to any user on a specific network, specify:

```
route add -net 192.100.201.0 192.100.13.7
```

- The 192.100.201.0 address is that of the receiving network (the `Destination` parameter).
- The 192.100.13.7 address is that of the routing network (the `Gateway` parameter).
- Example 3:

To establish a default gateway, specify:

```
route add 0 192.100.13.7
```

The value 0, or the default `ph`, for the `Destination` parameter indicates that any packets sent to destinations that were not previously defined and not on a directly connected network, go through the default gateway. The 192.100.13.7 address is that of the gateway chosen to be the default.

- Example 4:

To clear the host gateway table, specify:

```
route -f
```

- Example 5:

To add a route specifying weight and policy information, specify:

```
route add 192.158.2.2 192.158.2.5 -weight 5 -policy 4
```

- Example 6:

To set the weight and policy attributes of a preexisting route, specify:

```
route set 192.158.2.2 192.158.2.5 -weight 3 -policy
```

Host Server Encryption tab

Encryption keys are used for encrypting sensitive data in configuration files, or message data in Cloverleaf. It is suggested that all Cloverleaf servers in a company use the same encryption keys.

Because encryption is required for security server, an **Encryption** tab is on the host server and security server tabs.

When the security mode is advanced security, **Synchronize Encryption Keys with Security Server** is enabled. Clicking **Synchronize** synchronizes the encryption key database between the host server and security server.

Import Encryption Keys from other Host Servers is for importing encryption keys from other host servers.

Click **Import** to open the Import **Encryption Keys** dialog box for entering the host server name or IP address with the RMI registry port. The port is optional. Select **Overwrite local configured keys** to update the active encryption keys in the local host server.

Click **OK** to start the importing process.

The **Generate** buttons are for generating and configuring the key. For example, clicking **Generate** for Configuration Files Key would generate a key and then configure it as a configuration files key. The text field would update with `Configured`.

For the remaining sections, see [Security Server Encryption tab](#).

Host Server LDAP tab

The host server authenticates users against users' certificate files and private key files in basic/advanced security mode. Users must provide their user name, password, and certificate files and private key files when logging on to the system host server.

With an integrated LDAP server, users can employ their accounts on the LDAP server to log in to the system host server. The system automatically manages the user's certificate file and private key file, which are required for establishing an SSL connection.

To access the host server, the system client application acquires the LDAP integration configuration from the host server by HTTPS.

- If LDAP integration is disabled, then the system client application attempts to establish an RMI connection. This is attempted over SSL to the host server with the user account, existing certificate file, and private key file. This is the original behavior.
- If LDAP integration is enabled, then the system client application sends the LDAP user account to the host server. The host server attempts to connect to the configured LDAP server using the user account. If the connection can be successfully established, then the system grants the user's access request.
- A connection is established and the user account is a member of a group. If the group is configured on the LDAP server, then the system ensures a valid certificate and private key files exist for the user on the client side.

If not, then these files are automatically created and are ready on the client side. After that, the client can establish an RMI connection over SSL with the user account, certificate file, and private key file.

Configuring LDAP authentication

To configure LDAP authentication, open the **Server Administration > Host Server > LDAP** tab.

- 1 Select **Enable LDAP Authentication** to enable the fields.

This enables/disables LDAP integration. When selected, all GUI components are enabled. This option is cleared by default.

When this is selected and a user logs on, their user name and password credentials are authenticated against the LDAP server.

If the LDAP server cannot be contacted, then the user is not authenticated and an error message is returned to the user, stating `Unable to contact LDAP server`.

If the LDAP server authenticates their credentials, then their group is then checked against the LDAP server.

If the group that is returned by the LDAP server matches one of the configured groups, then the user is granted access. The user's access then follows the parameters configured in the user's profile.

If the user credentials do not authenticate or the group does not match, then the user does not have access. An error message is returned to the user.

2 Specify a host name in the **Host Name** field.

Specify the LDAP server's host name or IP address. You also have the option of specifying a backup LDAP server in cases where the primary server is down or cannot be reached. To use a backup server, specify the original and backup servers, separated by a comma.

3 Optionally, you can specify a port in the **Port** field.

If unspecified, then use 389 for the **No Encryption** and **StartTLS Extension** encryption methods, and 636 for the **SSL Encryption** method.

4 For **Encryption Method**, select an encryption method from the menu. The default is **SSL Encryption**.

5 For **Authentication Method**, select an authentication method from the menu.

Select from **Simple Authentication OR GSSAPI (Kerberos)**.

The Kerberos service is a client-server architecture that provides secure transactions over networks. This service offers strong user authentication, integrity, and privacy.

A user begins a Kerberos session by requesting a ticket-granting ticket (TGT) using the username/password from the Key Distribution Center (KDC). The TGT is similar to a passport, where, the ticket-granting ticket identifies you. Then, you can obtain numerous "visas." The "visas" (tickets) are not for foreign countries but for remote machines or network services. The KDC accesses a database to authenticate your identity. It then returns a ticket that grants you permission to access the other machine or services such as the IDE.

When **GSSAPI (Kerberos)** is selected, additional fields open for configuration.

- For **Kerberos Realm** specify the logical network, similar to a domain, that defines a group of systems that are under the same main KDC. Usually, the realm name is the same as your DNS domain name, except that the realm name is in uppercase.
- For **KDC Host**, specify the host name of the KDC (Kerberos Distribution Center) server.
- For **KDC Port**, specify the port number of the KDC server.

This information is located in the `krb5` file, which is the Kerberos configuration file, and is found in one of these locations:

- `/etc/krb5/krb5.conf` (Solaris)
- `C:\winnt\krb5.ini` (Windows)
- `/etc/krb5.conf` (Linux)

Note: **Manager Distinguished Name** in advanced settings is not supported when **GSSAPI (Kerberos)** is selected.

6 For **Default Domain Name**, specify the default domain name for the user account name. The default domain name is appended to the user name if it does not already use it. This is not required.

For example, a full account name would be harry@infor.com. If the user only specifies "harry," then it appends "@infor.com" to make the full account name to log in to the LDAP server.

- 7 Click **Test** to open the **Authentication Testing** dialog box. This tests if the host server can connect to the configured LDAP server.
OK is enabled when **User Name** and **Password** are specified, and is disabled when one is empty. Click **OK** to start the connecting process to the LDAP Server.
- 8 Click **Advanced** to open the **LDAP Advanced Configuration** dialog box.
 See [LDAP Advanced Configuration dialog box](#).

LDAP Advanced Configuration dialog box

This dialog box is for configuring parameters which are used for LDAP query. The parameters can be different for different LDAP servers, such as Sun ONE Directory server, Open LDAP Server, and so on.

Some configuration items address other LDAP v3 compliant servers, such as Open LDAP. Unlike Active Directory Server, where you use a user ID to log in, these LDAP servers require users to use a distinguished name to log in.

In this way, it uses a helper account to resolve the user's distinguished name. Therefore, the user can still use a user ID when logging on to the host server. The host server first resolves the user's distinguished name. Then the host server logs on to the LDAP Server with the resolved distinguished name and user password.

This table shows the available parameters:

Parameter	Description
Attribute for User ID	The attribute name of the user account on the LDAP server. For Active Directory, this is sAMAccountName.
User Classes	The class names for user entries on the LDAP server.
Group Classes	The class names for the group entry on the LDAP server.
Root Naming Contexts	The name of the root entry. All other entries, including user and group entries, are under the root entry. For Active Directory, there is a rootDomainNamingContext attribute that specifies the root naming context; for other LDAP servers, no such attribute exists. Specify the attribute for other LDAP servers, or leave blank for Active Directory.
User Search Base	A node's distinguished name that is defined in the LDAP server, under which all users are defined.
Manager Distinguished Name	The distinguished name of a helper user, which is used for resolving the other user's distinguished name.
Manager Password	The password for the LDAP server for the above user.

Groups

The group list is for configuring LDAP groups. A user account can access the specified host server only if that account is a member of any of the defined groups on the list.

If the group list is empty, then any user account can access the host server.

Any group that is configured on the group list should already be configured on the LDAP server before the configuration. This tool does not effect any changes on the LDAP server.

This table shows the available actions:

Action	Description
Add	<p>This adds a new group to the group list. Clicking this prompts a dialog box for specifying a group name. Clicking OK on the dialog box adds the new group into the group list, and that new group becomes selected.</p> <p>If a group with the same name already exists in the group list, then no new group is added. The group with the same name becomes selected.</p>
Edit	<p>This edits an existing group name. Clicking this prompts a dialog box for editing a group name. Clicking OK updates the group's name. If the new name is the same as that of another group, then the current group is removed. The group with the new name becomes selected. This is enabled if one and only one group is selected.</p>
Remove	<p>This removes selected groups from the group list. Clicking this removes the selected groups immediately. This is enabled if one or more groups are selected.</p>
Import	<p>This imports existing groups defined on the configured LDAP server. Clicking this prompts a dialog box for specifying the user name and password for logging on to the LDAP server. This is only if they have not been specified in the current application session. After this, it connects to the LDAP server and retrieves the LDAP groups from the LDAP server.</p> <p>A Choose LDAP Groups dialog box opens for selecting the LDAP groups.</p>

Account exception list

On the **Exceptions** tab, if **Enable user account exceptions** is cleared (default), then all user accounts must be authenticated against the LDAP server. If a user account fails to be authenticated, then access is denied to the user's account.

If the check box is selected and a user account is on the exception list, then the user account is not authenticated against the LDAP server. Instead, the system authenticates the user account with the existing certificate file and private key file. Certificate files and private key files for these user accounts are not maintained on the exception list.

LDAP exceptions support the regular exception rule to add users to the exception list.

For example, `^infor[-].*` adds LDAP exceptions for `infor-admin`, `infor-mdrown`, and `infor-votrain`.

Create user certificate file and private key file automatically

This check box is for specifying whether the system should create a user certificate file and private key file automatically for LDAP user accounts. This is selected by default.

If this check box is selected and no valid certificate and private key files exist on the client side, then a user certificate and private key file are automatically created for valid user accounts.

When the check box is selected, saving a configurations prompts a dialog box for specifying the CA password. This is only if the host server does not know the valid CA name and password. These credentials are required during the issuance of the certificate file. It saves the encrypted password into `ldap.xml`.

When this check box is cleared, the administrator must issue a valid certificate file and private key file for a user account. This must be completed before the user account can access the system host server.

Host Server Launch Bar Tools tab

This tab gives you the option of selecting which Launch Bar tools to hide/show. This is configured using the **Visible** check box for that tool. When a configuration tool is configured as “invisible”, the tool does not display on the IDE. The related Testing Tool is also invisible.

For example, to provide a user interface for healthcare domains, you could hide Edifact, VRL, FRL, NCPDP, HPRIM, HRL, and X12. NCPDP is optional, as this is related to Pharmacy.

User exception List

In basic and advanced security, a User Exception List pane is available where you can restrict which users can access a tool.

You can add an exception user list for each tool that is visible. For example, if a tool is available, but has an exception user list of "UserA, UserB", those users cannot see the tool.

To add users to the list:

- 1 Highlight a tool that is visible.
- 2 Click **Add** to open the **Add User Exception** dialog box. The list contains users with a valid certificate in the host server. Add users as required.

Host Server Version Control tab

In some situations, the administrator must directly revise the configuration file's version status. For doing this, the Server Administration tool has a **Version Control** tab to list version records. Users can update the version status in this tab.

- Selecting **Root** lists the root-level version records.
- Selecting **Site** enables the site name list. When a user selects a site from the list, the tab shows the specified site's version records.

The administrator can revise the configuration file's version status by these actions:

- Selecting **Unlock** to unlock the selected configuration entries.
- Selecting **Remove** to remove the selected configuration entries. All history version records of the selected configurations are removed from the version database and folder.
- Selecting **Refresh** to reload the table components.

Host Server Databases tab

Cloverleaf administrators can configure encryption settings for the SMAT, Recovery, and Error Databases for a selected site on the **Server Administration > Host Server > Databases** tab.

This tab has a list of available sites. Selecting a site enables the encryption options for that site. Configured options are stored at the site level.

In no security or basic security, you can change the database's encryption settings.

In advanced security, for the currently selected site, if you have permission to log in to **Server Administration**:

- An error message opens when you do not have read permission for the database encryption configuration file.
- If you have read permission for the database's encryption configuration file, then the UI for database encryption in the **Server Administration** is refreshed. This action updates the current database's encryption settings for the currently selected site.

If you also have write permission on the database's encryption configuration file, then you can click **Save**. Otherwise, an error message opens notifying that the save action failed.

All database encryption settings are stored in the `HCISITEDIR\siteSecurityInfo` encrypted file.

In each section of the **Host Server** tab, there are options to **Disable/Enable** the database, and to change the **Password**. This is available only when encryption is enabled.

This table lists the available options:

Option	Description
Site	The site selection menu is populated with a list of all sites in the root, and includes a No Sites option.
Internal Database Options	<p>For Data Encryption, when Enable is selected, all internal databases are encrypted.</p> <p>You can change the internal database password by clicking Change Password. When the field is empty, the default key is used. Newly created sites use this setting by default.</p> <p>When this is disabled, the internal database stores messages in plain text. Otherwise, it is password protected. In this case, <code>internaldbkey</code> is used for the internal database.</p>
Error Database Options	<p>For Data Encryption, when Enable is selected, the error database is encrypted.</p> <p>You can change the error database password by clicking Change Password. When the field is empty, the default key is used. Newly created sites use this setting by default.</p> <p>When this is disabled, the error database stores messages in plain text. Otherwise, it is password protected. In this case, <code>errordbkey</code> is used for the error database.</p>
SMAT Database Options	<p>When Save into Database is selected, SMAT messages are saved to the database. Otherwise, messages are saved into the <code>.idx</code> and <code>.msg</code> files.</p> <p>Selecting Save into Database also enables Data Encryption.</p> <p>For Data Encryption, when Enable is selected, the SMAT database is encrypted.</p> <p>If no SMAT database password is specified, then the site name is used as the default password. Newly created sites use this setting by default.</p>

Example: Enabling site database encryption

A new lab interface has been designed and deployed into a new site 'lab2'.

Encryption is enabled on the SMAT database.

After logging into the **Server Administration** tool, the **Databases** tab is selected. Then, the site 'lab2' is selected from the site list.

Encryption is enabled for the SMAT, Recovery, and Error databases and the encoding password is set for the selected site (lab2).

A custom password can now be set when enabling encryption on each object.

Host Server Certificate tab

For administrators, when changing passwords in Certificate Manager it is unnecessary to have the old password.

On this tab, administrators can change a password without knowing the previous password.

By default, this option is not selected.

The default password rule description is:

- Passwords must have 8-15 characters.
- Passwords must use uppercase and lowercase characters.
- Passwords must have at least one digit.
- Passwords must not contain the user name.

The password is validated against the strength rule. If it fails, then a prompt is given saying that it does not match the rule description.

Note: CIS cannot be logged on when a user modifies the password rule, but does not know how to set the password to match the updated rule. In this case, use the default `hssecurity.ini` rule to cover the current one. However, by doing this all updated settings in `hssecurity.ini` are omitted. For example, security mode.

Allowlist tab

Cloverleaf provides full control and configuration of command actions, Java Driver protocols, and Java UPoCs that are available during engine and UI operation. The default list contains commands and jars that are commonly used to manage Cloverleaf. The Security Administrator must configure and approve any additional application commands, jars, or classes.

The allowlist has root, master site, and site levels. The root level allowlist contains Cloverleaf defined commands and jar files that are invisible and cannot be modified. This can be used by all sites. The allowlist defined by the master site can also be used by the runtime site.

The SHA-256 hash of the file/binary or class/jar file is stored with the allowlist entry. The hash is automatically updated when it is added to the allowlist. Only the files/binaries or classes/jars that have the matching hashes can be run.

The allowlist is configured at **Server Administration > Host Server**.

The **Allowlist** tab's "read"/"write" permission is controlled by ACLRoleMgr with "config/allowlist" at the root level.

- Users with "read" permission can view the tab.
- Users with "write" permission can edit the allowlist.

Cloverleaf commands that are issued through the Remote Command tool and IDE are predefined in the allowlist. You can add, edit, and remove user-defined commands using the **Allowlist** tab. This tab is controlled by the config/allowlist of the root level in Advanced Security mode. Users who have "read" permission can see this tab.

Commands that run are expanded to the full path according to the OS environment when it only has the command name. The commands are validated by both the file path and digest. The file path and digest should be distinct for each entry.

Cloverleaf shipped jars that are used in Java Driver protocols and Java UPoCs are predefined in the allowlist. You can add, edit, and remove Java classes/jars using the **Allowlist > Java** tab. Java classes/jars are validated by the file name and digest.

The name and digest cannot be duplicates when adding the java class/jar.

You also can update the hashes in the allowlist by clicking **Update All Hashes** when there are changes to the files/binaries or classes/jars.

Note: The default allowlist is invisible and cannot be modified. Only the user-defined allowlist's `Note` attribute can be edited. The master site allowlist is loaded by default. Any command or jar/class file is considered valid when it resides in the master site.

The Host Server and MonitorD validate that any remote command entered by the user is approved before running. Unauthorized commands are denied and the attempt is logged in the audit log. Only failed commands are logged in `server/logs/allowlist.log`.

User interface

You can add and remove user-defined commands or java classes/jars in a list.

Select **Add** to browse for a file/binary or class/jar. Then, you can add a note to describe the file/binary or class/jar to run.

The **Command** tab list is referenced wherever remote commands are required. This includes the Remote Commands tool and the `exec` alert action in the **Alert Configurator**.

Commands configured in the **Alert Configurator** are validated before being saved.

The **Java** tab list is referenced by Java Driver protocols and Java UPoCs. Unauthorized classes and jars are not added to class path.

Predefined and user-defined allowlists

For a predefined allowlist, commands with these extensions under the listed folders can be run:

- Windows extensions

- exe
- bat
- cmd
- Linux/UNIX extensions
 - pl
 - sh
 - htc
 - no extension
- Folders
 - %HCIR00T%/bin
 - %HCIR00T%/sbin
 - %HCIR00T%/clgui/bin
 - %HCIR00T%/clgui/java/bin
 - %HCIR00T%/tcl/bin
 - %HCIR00T%/clgui/bin/misc

User-defined allowlist

The **Server Administration > Host Server** tab has a **Allowlist** tab. This is where you can create a user-defined allowlist. You must select a site before configuring it, since the user-defined allowlist is site-specified.

The **ACL Role Manager** has an `allowlist` node under the `config` node at the root level. This controls the permissions of allowlist access through CLAPI and **Server Administration**.

You must restart the Host Server after removing allowlist commands if the CLAPI is not enabled. This is because the host server and **Server Administration** are not in the same JVM.

Allowlist commands are stored in an encrypted SQLite database. The `allowlist.db` database file is installed under `%HCIR00T%/siteName/conf/`.

Command validation

The allowlist validates remote command running. The commands defined in the master site and root level are also allowed in the current site.

- Only remote commands are validated. Internal commands and commands not issued through an RMI invocation can be bypassed.
- An error message opens when a command fails on allowlist validation.
- All validation results are recorded in `server\audit\audit.log`. Failed results are logged in `server\logs\allowlist.log` for troubleshooting.

Commands that are configured in the Alert Configurator are validated before being saved. The runtime alert exec validation is through `hcimonitor.d`.

Java validation

The CIS engine performs an allowlist check on all path elements that are passed to it as Java Classpath when starting an embedded JVM for Java Driver and Java UPoC. On the server, the CIS engine composes a JVM Classpath with these elements:

- User-assigned classpath elements from Java Driver configuration. This is the `CLASSPATH` property in `pni` files.
- All jar files directly under `HCIROOT/lib/java`.
- All jar files directly under `HCIROOT/java_uccs`.
- All jar files directly under `HCISITEDIR/java_uccs`.
- All elements in the environment variable `CLASSPATH` in the terminal context where an engine process starts. This includes at least:
 - `HCIROOT/clgui/lib/cljava.jar`.
 - `HCIROOT/python/jep/jep-*.jar`.
 - All classes under `HCIROOT/java_uccs`.
 - All classes under `MASTERSITEDIR/java_uccs`.
 - All classes under `HCISITEDIR/java_uccs`.
 - All classes under JVM working directory, which is the `startdir` in the `pni` file if set or `HCISITEDIR/exec/processes/PROCESS_NAME`.

All existing paths from this list are verified by comparing SHA digest to that saved in Java allowlist. In most cases, element paths are resolved to absolute paths by the engine.

Using CLAPI in allowlist management

There are three main steps for adding a new command and confirming that it has been successfully added.

- 1 Acquire the CSRF token before adding a new command into the allowlist.

On the command line, run:

```
curl -G https://127.0.0.1:15067/clapi/api/security/csrf -k
--header "Authorization:Basic YWRtaW5pc3RyYXRvcjpQQHNzd29yZDAx" -c cookie.out -v
```

Results:

- There is feedback on the command line.
 - The file `cookie.out` is created.
 - There is feedback “{ "csrf": "04c0d8c5-de00-4f3c-bffb-a7ef292921ef" }HTTP/1.1 200 OK” on the command line.

Note: The CSRF token session is different for each call. You must replace the next step's CSRF token session of cases with the actual one.
- 2 Update the allowlist. There are four APIs for updating the command/commands. Replace the `{siteName}` with a real site name such as “helloworld”.

- To update the command allowlist entries, on the command line, run:

```
curl https://127.0.0.1:15067/clapi/v3/api/admin/server-admin/{siteName}/allowlist-config
-k
--header "Authorization:Basic YWRtaW5pc3RyYXRvcjpwQHNzd29yZDAx"
--header "X-CSRF-TOKEN: 04c0d8c5-de00-4f3c-bffb-a7ef292921ef" -b cookie.out -d @$SITE
DIR/test/
serveradmin_setAllowListConfig.json --request POST -v
```

This results in “HTTP/1.1 200 OK” feedback on the command line.

If you require adding the `ls` command, then the file content in the `$$SITE$DIR/test/serveradmin_setAllowListConfig.json` file is:

```
{"entries": [{"name": "ls.exe", "note": "this is note for command ls", "path":
"C:\\cygwin\\bin\\ls.exe"}, {"name": "ps.exe", "note": "this is note for command ps", "path":
"C:\\cygwin\\bin\\ps.exe"}]}
```

- To add a new command, on the command line, run:

```
curl https://127.0.0.1:15067/clapi/v3/api/server-admin/{siteName}/allowlist-config/en
tries?type=command
-k --header "Authorization:Basic YWRtaW5pc3RyYXRvcjpwQHNzd29yZDAx" --header "X-CSRF-TOKEN:
04c0d8c5-de00-4f3c-bffb-a7ef292921ef" -b cookie.out -H "Content-Type:application/json"
-d @$HCISITEDIR/test/serveradmin_addAllowlistEntry.json --request POST -v
```

This results in “HTTP/1.1 200 OK” feedback on the command line.

If you require adding the `ls` command, then the file content in the `$$SITE$DIR/test/serveradmin_addAllowlistEntry.json` file is:

```
{"name": "ls.exe", "note": "this is note for command ls", "path":
"C:\\cygwin\\bin\\ls.exe"}
```

- To update the node of a command, run:

```
curl https://127.0.0.1:15067/clapi/v3/api/server-admin/{siteName}/allowlist-config/en
tries/{entryId}
?type=command -k --header "Authorization:Basic YWRtaW5pc3RyYXRvcjpwQHNzd29yZDAx"
-d @$HCISITEDIR/test/serveradmin_updateAllowlistEntry.json --header "X-CSRF-TOKEN:
04c0d8c5-de00-4f3c-bffb-a7ef292921ef" -b cookie.out --request PUT -v
```

This results in “HTTP/1.1 200 OK” feedback on the command line.

If you require updating the note of the `ls` command, then the file content in the `$$SITE$DIR/test/serveradmin_updateAllowlistEntry.json` is:

```
{"note": "this is updated note for command ls"}
```

- To delete a command, run:

```
curl https://127.0.0.1:15067/clapi/v3/api/server-admin/{siteName}/allowlist-config/en
tries/{entryId}
?type=command -k --header "Authorization:Basic YWRtaW5pc3RyYXRvcjpwQHNzd29yZDAx" --header
"X-CSRF-TOKEN:
04c0d8c5-de00-4f3c-bffb-a7ef292921ef" -b cookie.out -H "Content-Type:application/json"
--request DELETE -v
```


This results in “HTTP/1.1 200 OK” feedback on the command line.

3 On the command line, run:

```
curl -G https://127.0.0.1:15067/clapi/v3/api/admin/server-admin/{siteName}/allowlist-config
-k
--header "Authorization:Basic YWRtaW5pc3RyYXRvcjpQQHNzd29yZDAX" -v
```

Results:

- On the command line, there is feedback: “HTTP/1.1 200 OK”..
- On the command line, there is feedback:

```
{"entries": [{"id": 1, "name": "ls.exe", "note": "this is note for command ls", "path":
"C:\\cygwin\\bin\\ls.exe", "digest":
"2427ae41e4649b934ca495991b7852b855e3b0c44298fc1c149afbf4c8996fb9"},
{"id": 2, "name": "ps.exe", "note": "this is note for command ps", "path": "C:\\\\cyg
win\\bin\\ps.exe", "digest":
"e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"}]}
```

Updating the hashes of a specific site

There is an API for updating all the hashes of a specified site.

On the command line, run:

```
curl https://127.0.0.1:15067/clapi/v3/api/server-admin/{siteName}/allowlist-config/hashes
-k --header "Authorization:Basic YWRtaW5pc3RyYXRvcjpQQHNzd29yZDAX" --header "X-CSRF-TOKEN:
04c0d8c5-de00-4f3c-bffb-a7ef292921ef" -b cookie.out --request PUT -v
```

Result:

There is “HTTP/1.1 200 OK” feedback on the command line.

Allowlist examples

Examples of setting up the allow list include:

- [Setting the java allowlist](#) on page 1131
- [Adding a "jar" file](#) on page 1133
- [Adding a remote command task](#) on page 1134

Setting the java allowlist

To set the java allowlist:

- 1 Acquire the CSRF token before adding a new command into the allowlist.

On the command line, run:

```
curl -G https://127.0.0.1:15067/clapi/api/security/csrf -k
--header "Authorization:Basic YWRtaW5pc3RyYXRvcjpQQHNzd29yZDAX" -c cookie.out -v
```

Results:

- There is feedback “HTTP/1.1 200 OK” on the command line.
- The file `cookie.out` is created.
- There is feedback on the command line.

```
{ "csrf": "04c0d8c5-de00-4f3c-bffb-a7ef292921ef" }
```

Note: The CSRF token session is different for each call. You must replace the next step's CSRF token session of cases with the actual one.

2 Update the allowlist. There are four APIs for updating the `jar/jars`.

- Update the entire java allowlist.

On the command line, run:

```
curl https://127.0.0.1:15067/clapi/v3/api/admin/{siteName}/server-admin/allowlist-config?type=java
-k --header "Authorization:Basic YWRtaW5pc3RyYXRvcjpQQHNzd29yZDAX" --header "X-CSRF-TOKEN:
04c0d8c5-de00-4f3c-bffb-a7ef292921ef" -b cookie.out -d @$SITEDIR/test/serveradmin_setAllowListConfig.json -v
```

This results in “HTTP/1.1 200 OK” feedback on the command line.

If you require adding the `ls` command, then the file content in `$$$SITEDIR/test/ serveradmin_setAllowListConfig.json` is:

```
{"entries": [{"name": "dt.jar","note": "this is note for jar dt","path":
"C:\\Program Files\\Amazon Corretto\\jdk1.8.0_252\\lib\\dt.jar"}]}
```

- Add a new `jar/class`.

On the command line, run:

```
curl https://127.0.0.1:15067/clapi/v3/api/server-admin/{siteName}/allowlist-config/entries?type=java -k --header "Authorization:Basic YWRtaW5pc3RyYXRvcjpQQHNzd29yZDAX"
--header "X-CSRF-TOKEN: 04c0d8c5-de00-4f3c-bffb-a7ef292921ef" -b cookie.out
-H "Content-Type:application/json" -d @$HCISITEDIR/test/serveradmin_addAllowlistEntry.json
--request POST -v
```

This results in “HTTP/1.1 200 OK” feedback on the command line.

If you require adding the `ls` command, then the file content in `$$$SITEDIR/test/serveradmin_addAllowlistEntry.json` is:

```
{"name": "dt.jar","note": "this is note for jar dt","path":
"C:\\Program Files\\Amazon Corretto\\jdk1.8.0_252\\lib\\dt.jar"}
```

- Update the node of a `jar/class`.

On the command line, run:

```
curl https://127.0.0.1:15067/clapi/v3/api/server-admin/{siteName}/allowlist-config/entries/{entryId}
?type=java -k --header "Authorization:Basic YWRtaW5pc3RyYXRvcjpQQHNzd29yZDAx"
-d @$HCISITEDIR/test/serveradmin_updateAllowlistEntry.json --header "X-CSRF-TOKEN:
04c0d8c5-de00-4f3c-bffb-a7ef292921ef" -b cookie.out --request PUT -v
```

This results in “HTTP/1.1 200 OK” feedback on the command line.

If you require updating the `ls` command note, then the file content in `$$SITEDIR/test/serveradmin_updateAllowlistEntry.json` is:

```
{"note": "this is updated note for dt.jar"}
```

- Delete a jar/class.

On the command line, run:

```
curl https://127.0.0.1:15067/clapi/v3/api/server-admin/{siteName}/allowlist-config/entries/{entryId}?type=java -k --header "Authorization:Basic YWRtaW5pc3RyYXRvcjpQQHNzd29yZDAx"
--header "X-CSRF-TOKEN: 04c0d8c5-de00-4f3c-bffb-a7ef292921ef" -b cookie.out
-H "Content-Type:application/json" -request DELETE -v
```

This results in “HTTP/1.1 200 OK” feedback on the command line.

3 Acquire the allowlist to confirm the jar has been successfully set.

On the command line, run:

```
curl -G https://127.0.0.1:15067/clapi/v3/api/admin/server-admin/{siteName}/allowlist-config?type=java -k --header "Authorization:Basic YWRtaW5pc3RyYXRvcjpQQHNzd29yZDAx"
-v
```

This results in “HTTP/1.1 200 OK” feedback on the command line.

Additional feedback is on the command line:

```
{"entries": [{"id": 1, "name": "dt.jar", "note": "this is note for jar dt", "path":
"C:\\Program Files\\Amazon Corretto\\jdk1.8.0_252\\lib\\dt.jar ", "digest":
"8fbf43eb41fe057bcad2be1f1f688f1c31d90ee08118eae26ca176d60cd9b69c"}]}
```

Adding a "jar" file

Note: You can use `jython.jar` for writing Python scripts.

To add a jar file:

- 1 Open the **Server Administration** and select the **Allowlist > Java** under the **Host Server** tab.
- 2 Click **Add**. This opens a dialog box with a **Browse** button. Click **Browse** to navigate to the `jython.jar` file. In **Note**, label the command "Java implementation for Python version".
- 3 Click **Save**. The java allowlist is updated and shows `jython.jar` as an available jar with an alias of "Java implementation for Python version".

Adding a remote command task

You can use `hciengineerun` and `hcienginestop` to manage processes in a shell session.

To add a remote command task:

- 1 Open the **Server Administration** dialog box and select the **Allowlist > Command** tab under the **Host Server** tab.
- 2 Click **Add**. This opens a dialog box with a **Browse** button. Click this to navigate to the `hciengineerun` command. In the note, label the command **Start the engine**.
- 3 Click **Save**. The command list is updated and shows `hciengineerun` as an available remote command with an alias of **Start the engine**.
- 4 Repeat the same steps to add `hcienginestop` with the label **Stop the engine** to the list of approved commands.

Security Audit tool

To document and evaluate the security risk of their integration implementation, Cloverleaf security administrators can use the Security Audit tool. This screens and evaluates the security risk that is associated with user developed/implemented Cloverleaf integration objects.

Note: For developers that create custom code connections to Cloverleaf that are hosted by Infor, security best practice is to first use the Audit Tool. Then, remediate any findings from the security audit.

The security audit options are accessed in the **Server Administration > Security Audit** tab.

With this tool, you can scan the Cloverleaf host servers' user-defined configurations in the currently selected root or site and generate a security audit report. The generated report is available in plain text. The default is a full instance report.

This report alerts you to potential security risks in High, Medium, and Low risk levels. Then, you can mitigate these vulnerabilities, based on the suggestions in the report.

Evaluated items include:

- Cloverleaf security level. An alert is prompted if advanced security is not enabled.
- UPOC scripts are scanned for system and exec calls.
 - TCL scripts are scanned for reserved words that are used in lines that are not comments. These include `open`, `read`, `mkdir`, `eval`, and `exec`.
 - Java class files are scanned.
- Protocols are evaluated to determine if they are configured to connect outside of the local network. Users are cautioned when connecting outside of their domain.
- Users are notified if they enable a transport encryption layer version that is not current. Users are cautioned if configuring a protocol to use a transport layer that is not the latest supported by Cloverleaf.
- Connections are tested with the connecting system. Users are alerted if a connecting system negotiates the transport layer version down.

- Users are cautioned on all protocol connections that are not encrypted. This check is disabled through an argument to the audit tool. If the user decides to disable this, then it is noted on the audit report. The audit report is encrypted and stored on the host server.

Server interface

The Security Audit tool can be run through a CL command that optionally accepts a site name and file name as parameters. By default, the security audit report is generated for all sites and HCIROOT.

Passing the site name parameter generates the security audit report for the designated site.

Passing a file name to the security audit tool results in the unencrypted reported being written to the file name for retrieval by the user.

Security Audit options

The Security Audit feature is accessed in the **Server Administration > Security Audit** tab.

This table lists the audit report options in the tab:

Option	Description
Scan Target	Specify the scanning sites in this field. Click Select to select from a list of targets. The IDE scans all sites and the root level. By default, this field is empty.
Output	Displays the report generation progress and command line output.
Clear	Cleans up the content for the current text in the output.
View	Opens the Security Audit Report Viewer dialog box. When the dialog box is opened, it attempts to merge the reports from the specified sites by default. If a site report does not exist, then the text area displays There is no security audit report for 'XX'..
Export	Opens a file browser, where you can select a file path to save the report to a plain text file. If the selected file already exists, then a message displays: Security Audit report file 'XXX' already exists. Do you require to overwrite it?. Clicking Yes overwrites the existing file. Clicking No exits the export function.
Generate	This runs a security scan. For each site, it generates a single site-level scan result file. See hcisecurityaudit command usage .

Option	Description
Cancel	The security audit scan is a time-consuming process. Clicking Cancel ends the current scan. For each site or root, when the scan action is finished, the scanner attempts to handle the <code>xml</code> file. It copies the scan result from memory to the <code>xml</code> file. If the user cancels the scan in the process, then a generated scan result file is kept.

Audit report

You can view the audit report in the Security Audit tool. A copy of the report can be downloaded to your desktop through the tool.

With the tool, you can generate a full instance report or a report limited to a specific site. The default is to generate a full instance report. After a report generation is started, it can be stopped if the generation is taking too long.

Generated reports can be viewed in the tool when the report generation is complete. The report is encrypted after the report is generated.

Selecting **Export** opens the **Save As** dialog box. In this dialog box, you can save a copy of the text report to the local disk. When doing this, a warning is prompts, stating that exporting as plain text may expose sensitive system vulnerability information.

The site-level scan result file is located in the `$HCIR00T/securityaudit/site name` folder.

The root-level scan result file is located in the `$HCIR00T/securityaudit` folder.

Audited items

This table lists the items that are audited by the **Security Audit** tool:

Item	Description
\$HCIR00T/server/ hssecurity.ini: Security mode	<p>Advanced security mode ensures that the maximum amount of defensive support is enabled within Cloverleaf. Basic security does mitigate some risk through user security that is enforced by certificates, but it is not the recommended level of security. You should ensure strong business needs are driving running Cloverleaf on basic security.</p> <p>Turning off security mode makes the environment more vulnerable to being hacked. You must ensure that the maximum amount of defensive support is enabled within Cloverleaf by turning on advanced security mode.</p> <p>Basic security mitigates some risk through user security that is enforced by certificates, but it is not the recommended level of security. You must ensure extremely strong business needs are driving running Cloverleaf with No security mode enabled.</p>
\$HCIR00T/server/ hssecurity.ini: Audit Server	<p>Turning off the Audit Server prevents oversight of activity on the system, which makes detection of a data breach and malicious behavior in general difficult to track. Ignorance of a data breach is not defense against liability for a data breach. You should strongly consider turning this feature back on.</p>
\$HCIR00T\tclprocs	<p>The system is vulnerable to command injection when a command can manipulate the host operating system or the file system files. Command injection is an attack in which the goal is running malicious commands on the host operating system.</p> <p>In this attack, the attacker-supplied operating system commands are usually run with the privileges of the vulnerable application.</p> <p>The best remedy is to discourage this functionality. If that is not possible, then you must validate all user-supplied input to ensure unintentional or malicious OS commands have no opportunity to run.</p>
\$HCISITE\ java_uccs	<p>The system is vulnerable to command injection when a command can manipulate the host operating system or the file system files. Command injection is an attack in which the goal is running malicious commands on the host operating system.</p> <p>In this attack, the attacker-supplied operating system commands are usually run with the privileges of the vulnerable application.</p> <p>The best remedy is to discourage this functionality. If that is not possible, then you must validate all user-supplied input to ensure unintentional or malicious OS commands have no opportunity to run.</p>

Item	Description
hcverify results	The integrity of the system is maintained by ensuring all aspects of the system are secure. Hackers seek out the weakest points in a system. Not enabling runtime and library integrity checking means that a malicious alteration of those libraries go unnoticed longer. This risks a breach going unnoticed for a long period of time.
Netconfig fileset-ftp/host	The best defense against malicious behavior is to allowlist what is permissible. Without limiting the permitted hosts, connections outside of the domain to any public place are possible. Limitations on allowed hosts should be configured as soon as possible.
Netconfig fileset-ftp/FTPS Secure Option TLSV1.1/1 + Any mode High risk	TLS v1.1 has known vulnerabilities which could be exploited to intercept encrypted communication. Without correct validation of the encryption cipher, you can spoof the chain of trust underlying the client's encryption token. Someone could see or change the information you send or get through this site. You should only support TLS v1.2 with full client authentication.
Netconfig fileset-ftp/FTPS Secure Option TLSV1.2 + Client mode Medium risk	When a self-signed certificate is accepted, it lacks correct validation of the encryption cipher. It is difficult for the server to know definitively if the encryption cypher is legitimate. Someone could see or change the information you send or get through this site. You should only support TLS v1.2 with full client authentication.
Netconfig http-client/HTTPS TLSV1.1/1 + Any mode High risk	TLS v1.1 has known vulnerabilities which could be exploited to intercept encrypted communication. Without correct validation of the encryption cipher, you can spoof the chain of trust underlying the client's encryption token. Someone could see or change the information sent or received through this functionality. You should only support TLS v1.2 with full client authentication.
Netconfig http-client/HTTPS TLSV1.2 + Client mode Medium risk	When a self-signed certificate is accepted, it lacks correct validation of the encryption cipher. It is difficult for the server to know definitively if the encryption cypher is legitimate. Someone could see or change the information sent or received through this functionality. You should only support TLS v1.2 with full client authentication.
Netconfig http-client/Proxy	Proxy host may be at outside of user domain. The best defense against malicious behavior is to allowlist what is permissible. Without limiting the permitted hosts, connections outside of the domain to any public place are possible. Limitations on permitted hosts should be configured as soon as possible.

Item	Description
Netconfig java/direct-retriever/POP3 retriever	<p>Host may be at outside of user domain.</p> <p>The best defense against malicious behavior is to allowlist what is permissible. Without limiting the permitted hosts, connections outside of the domain to any public place are possible.</p> <p>Limitations on permitted hosts should be configured as soon as possible.</p>
Netconfig java/direct-retriever/SSL Socket Factory	<p>SSL Socket Factory does not support TLS v1.2. Prior versions of TLS have known vulnerabilities which could be exploited to intercept encrypted communication. Someone could see or change the information sent or received through this functionality.</p> <p>Do not use this functionality.</p>
Netconfig java/direct-sender/SMTP Sender	<p>Host may be at outside of user domain.</p> <p>The best defense against malicious behavior is to allowlist what is permissible. Without limiting the permitted hosts, connections outside of the domain to any public place are possible. Limitations on permitted hosts should be configured as soon as possible.</p>
Netconfig java/direct-sender/ SSL Socket Factory	<p>SSL Socket Factory does not support TLS v1.2. Prior versions of TLS have known vulnerabilities which could be exploited to intercept encrypted communication. Someone could see or change the information sent or received through this functionality.</p> <p>Do not use this functionality.</p>
Netconfig java/ion-retriever/ION Retriever	<p>Host may be at outside of user domain.</p> <p>The best defense against malicious behavior is to allowlist what is permissible. Without limiting the permitted hosts, connections outside of the domain to any public place are possible.</p> <p>Limitations on allowed hosts should be configured as soon as possible.</p>
Netconfig java/ion-sender/SMTP Sender	<p>Host may be at outside of user domain.</p> <p>The best defense against malicious behavior is to allowlist what is permissible. Without limiting the permitted hosts, connections outside of the domain to any public place are possible. Limitations on permitted hosts should be configured as soon as possible.</p>

Item	Description
Netconfig java/ws-client/Conduit	<p>Proxy server may be at outside of user domain.</p> <p>The best defense against malicious behavior is to allowlist what is permissible. Without limiting the permitted hosts, connections outside of the domain to any public place are possible.</p> <p>Limitations on permitted hosts should be configured as soon as possible.</p> <p>This functionality does not support TLS v1.2. Prior versions of TLS have known vulnerabilities which could be exploited to intercept encrypted communication. Someone could see or change the information sent or received through this functionality.</p> <p>Do not use this functionality.</p>
Netconfig java/ws-client/Soap Consumer	<p>Policy Generator/Use Transport Security is turned off. The best defense against malicious behavior is to allowlist what is permissible. Without limiting the permitted hosts, connections outside of the domain to any public place are possible.</p> <p>Limitations on permitted hosts should be configured as soon as possible.</p> <p>This functionality does not support TLS v1.2. Prior versions of TLS have known vulnerabilities which could be exploited to intercept encrypted communication. Someone could see or change the information sent or received through this functionality.</p> <p>Do not use this functionality.</p>
Netconfig java/ws-rawclient/Conduit	<p>Proxy server may be at outside of user domain. The best defense against malicious behavior is to allowlist what is permissible. Without limiting the permitted hosts, connections outside of the domain to any public place are possible.</p> <p>Limitations on permitted hosts should be configured as soon as possible.</p> <p>This functionality does not support TLS v1.2. Prior versions of TLS have known vulnerabilities which could be exploited to intercept encrypted communication. Someone could see or change the information sent or received through this functionality.</p> <p>Do not use this functionality.</p>

Item	Description
Netconfig java/ws-server/Engine	<p>Host may be at outside of user domain. The best defense against malicious behavior is to allowlist what is permissible. Without limiting the permitted hosts, connections outside of the domain to any public place are possible.</p> <p>Limitations on permitted hosts should be configured as soon as possible.</p> <p>This functionality does not support TLS v1.2. Prior versions of TLS have known vulnerabilities which could be exploited to intercept encrypted communication. Someone could see or change the information sent or received through this functionality.</p> <p>Do not use this functionality.</p>
Netconfig java/ws-server/SoapProvider	<p>Policy Generator/Use Transport Security is turned off.</p> <p>This functionality does not support TLS v1.2. Prior versions of TLS have known vulnerabilities which could be exploited to intercept encrypted communication. Someone could see or change the information sent or received through this functionality.</p> <p>Do not use this functionality.</p>
Netconfig mqs	<p>Server name may be at outside of user domain. The best defense against malicious behavior is to allowlist what is permissible. Without limiting the permitted hosts, connections outside of the domain to any public place are possible.</p> <p>Limitations on permitted hosts should be configured as soon as possible.</p>
Netconfig pdl-tcpip	<p>This is a depreciated functionality. It is recommended to discontinue use as soon as possible.</p> <p>Host may be at outside of user domain. The best defense against malicious behavior is to allowlist what is permissible. Without limiting the permitted hosts, connections outside of the domain to any public place are possible.</p> <p>Limitations on permitted hosts should be configured as soon as possible.</p> <p>SSL is turned off. This functionality does not support TLS v1.2. Prior versions of TLS have known vulnerabilities which could be exploited to intercept encrypted communication. Someone could see or change the information sent or received through this functionality.</p> <p>Do not use this functionality.</p>
Netconfig tcpip TLSV1.1/1 + Any mode High risk	<p>TLS v1.1 has known vulnerabilities which could be exploited to intercept encrypted communication. Without correct validation of the encryption cipher, you can spoof the chain of trust underlying the client's encryption token. Someone could see or change the information sent or received through this functionality.</p> <p>You should only support TLS v1.2 with full client authentication.</p>

Item	Description
Netconfig tcpip TLSV1.2 + Client mode Medium risk	<p>When a self-signed certificate is accepted, it lacks correct validation of the encryption cipher. It is difficult for the server to know definitively if the encryption cypher is legitimate. Someone could see or change the information sent or received through this functionality.</p> <p>You should only support TLS v1.2 with full client authentication.</p>
Netconfig tcpip	<p>Host may be at outside of user domain. The best defense against malicious behavior is to allowlist what is permissible. Without limiting the permitted hosts, connections outside of the domain to any public place are possible.</p> <p>Limitations on permitted hosts should be configured as soon as possible.</p> <p>SSL is turned off. This functionality does not support TLS v1.2. Prior versions of TLS have known vulnerabilities which could be exploited to intercept encrypted communication. Someone could see or change the information sent or received through this functionality.</p> <p>Do not use this functionality.</p>
Netconfig tcpip TLSV1.1/1 + Any mode High risk	<p>TLS v1.1 has known vulnerabilities which could be exploited to intercept encrypted communication. Without correct validation of the encryption cipher, you can spoof the chain of trust underlying the client's encryption token. Someone could see or change the information sent or received through this functionality.</p> <p>You should only support TLS v1.2 with full client authentication.</p>
Netconfig tcpip TLSV1.2 + Client mode Medium risk	<p>When a self-signed certificate is accepted, it lacks correct validation of the encryption cipher. It is difficult for the server to know definitively if the encryption cypher is legitimate. Someone could see or change the information sent or received through this functionality.</p> <p>You should only support TLS v1.2 with full client authentication.</p>
Site Preference SMAT/Data Encryption ((HIGH))	<p>Encrypting data that contains sensitive information is an effective control against malicious alterations and inappropriate access. The SMAT database is known to contain sensitive information. Turning off this control increases the risk of a data breach.</p> <p>Enabling encryption has a small effect on the performance of the system. This should be enabled.</p>
Site Preference Error Database Encryption ((Medium))	<p>Encrypting data that contains sensitive information is an effective control against malicious alterations and inappropriate access. Although the error database is unlikely to contain sensitive information, there is no guarantee that it never will. Turning off this control increases the risk of a data breach. Enabling encryption has a small effect on the performance of the system. This should be enabled.</p>

Item	Description
Site Preference Internal Database Encryption (Medium)	<p>Encrypting data that contains sensitive information is an effective control against malicious alterations and inappropriate access. Although the internal database is unlikely to contain sensitive information, there is no guarantee that it never will. Turning off this control increases the risk of a data breach.</p> <p>Enabling encryption has a small effect on the performance of the system. This should be enabled.</p>
\$HCISITE tclprocs folder Tcl file	<p>Permitting a command to manipulate the host operating system or the file system files makes the system vulnerable to command injection. Command injection is an attack in which the goal is running malicious commands on the host operating system. In this attack, the attacker-supplied operating system commands are usually run with the privileges of the vulnerable application. The best remediation is to not permit or encourage this functionality.</p> <p>If that is not feasible, then you must validate all user-supplied input to ensure unintentional or malicious OS commands are not permitted.</p>
\$HCISITE java_uccs Java class file	<p>Permitting a command to manipulate the host operating system or the file system files makes the system vulnerable to command injection. Command injection is an attack in which the goal is running malicious commands on the host operating system. In this attack, the attacker-supplied operating system commands are usually run with the privileges of the vulnerable application. The best remediation is to not permit or encourage this functionality. If that is not feasible, then you must validate all user-supplied input to ensure unintentional or malicious OS commands are not permitted.</p>
SSL cipher algorithm check For AES 64, 56, 28, and so on. For SHA1	<p>Although the Advanced Encryption Standard (AES) is trusted by the U.S. Government and numerous other organizations, using a 64-bit and lower key makes the cipher vulnerable to brute force attacks. A determined actor could crack the encryption in a reasonable time frame to make using 64-bit or lower keys a known vulnerability.</p> <p>It is recommended that only AES-128 is used.</p> <p>SHA-1 is not secure and extremely vulnerable against attacks by a determined actor. Many organizations no longer accept SHA-1 SSL certificates. For example, Microsoft, Google, and Apple.</p> <p>It is highly recommended that you instead use a minimum of SHA-2.</p> <p>For example, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256.</p>

Item	Description
CA root CA Root algorithm check	<p>Detects the CA certificate that uses the SHA-1 signature algorithm that is not recommended.</p> <p>SHA-1 is not secure and is extremely vulnerable against attacks by a determined actor. Many organizations no longer accept SHA-1 SSL certificates. For example, Microsoft, Google, and Apple. It is highly recommended that you instead use a minimum of SHA-2.</p> <p>For example, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256.</p> <p>Upgrade the CA certificate using the Certificate Manager tool.</p>

hcisecurityaudit command usage

The `hcisecurityaudit` command supports security audit result data generation for a specific site and root.

```
hcisecurityaudit [-t <site1,site2,...>] [-e [-d <export report path>]
[-f]] [-l] [-h | -help]
```

- `-t <site1,site2,...>`. For site, this is the name list of sites. For root, this is `HCIRoot`. Without this option, the root and all sites are scanned.
- `-d export report path` specifies the file path to export the security audit report.
If the path is a directory, then the default file name is `securityaudit_report.txt`.
- `-e` merges all scan results of specific sites into one plain text file.
- `-f` forces an overwrite of the export file.
- `-l` prints the scan process information to a log file.
- `-h` or `-help` displays this help message.

hciverify command usage

`hciverify` checks the `HCIRoot` runtime and library integrity, `rootInfo`, and `siteInfo`.

For this command to work, you must `setroot` before verifying.

```
hciverify [-v] [-s sites[,]] [-d] [-h]
```

- `-v` performs verification on `HCIRoot`.
- `-s sites[,]` lists the sites to verify.
- `-d` is verbose mode.
- `-h` displays this help. For this option to work, you must `setroot` into the root to verify.

`hciverify` checks:

- `rootInfo`

- Site that is defined in `rootInfo`. This is the site directory and `siteInfo` file.
- Master site that is defined in `rootInfo`. This has multi-mastersite support, and is backward compatible.
- Master site that is defined in the site.
- Cloverleaf binaries and libraries under `HCIRoot`.

Example hcverify report for security audit

```
C:\hcverify (good case)
Everything looks good.
```

Report examples

Note: These are only examples.

These examples are from root level and site level reports.

Root level

Security audit report for Cloverleaf `ROOT`.

Summary:

Severity	Number of alerts
High	2
Medium	0
Low	0

Server INI:

Classification	Severity	Key	Description
DEFINITIVE	HIGH	audit_server_used	<p>Turning off the Audit Server prevents oversight of activity on the system. This makes detection of a data breach and malicious behavior difficult to track.</p> <p>Ignorance of a data breach is not defense against liability for a data breach. Strongly consider turning this feature back on.</p>

Classification	Severity	Key	Description
DEFINITIVE	HIGH	seciurity_server_used basic_security_enabled	<p>Cloverleaf is working on None Security mode. Advanced Security mode ensures that the maximum amount of defensive support is enabled within Cloverleaf.</p> <p>Basic Security does mitigate some risk through user security enforced by certificates, but it is not the recommended level of security. Ensure strong business needs are driving the running of Cloverleaf on Basic Security.</p> <p>Ensure that the maximum amount of defensive support is enabled within Cloverleaf by turning on Advanced Security mode.</p>

Site level

Security audit report for site hvshd04.

Summary:

Severity	Number of alerts
High	11
Medium	0
Low	0

TCL scripting

Mitigation:

Permitting a command to manipulate the host operating system, or files on the file system, makes the system vulnerable to command injection. Command injection is an attack in that the goal is running malicious commands on the host operating system. In this attack, the attacker-supplied operating system commands are usually run with the privileges of the vulnerable application. The best remediation is to not permit or encourage this functionality. If this is not feasible, then all user-supplied input needs to be validated to ensure unintentional or malicious OS commands are not permitted.

Classification	Severity	Path	line	Description
SUSPECT	HIGH	C:\191r2\cis19.1\integrator\hvshd04\tclprocs\clud.tcl	254	The delete method is called.
SUSPECT	HIGH	C:\191r2\cis19.1\integrator\hvshd04\tclprocs\clud.tcl	259	The delete method is called.
SUSPECT	HIGH	C:\191r2\cis19.1\integrator\hvshd04\tclprocs\clud.tcl	305	The delete method is called.

Security Audit example

A user has completed development of several new interfaces. To ensure that the new interfaces do not increase the security exposure of the IT department, a site report can be created.

- 1 Open the **Server Administration > Security Audit** tab.
- 2 Selects any site, and then click **Generate**.
You can also create a root report by selecting `HCIR00T` and clicking **Generate**.
- 3 When the report generation is finished, you can view the report with the **Security Audit** tool.

System ports

The host server and RMI ports are per-root (that is, there is one for the whole system installation). These ports are primarily used by the GUI.

Fixed protocol ports are the port numbers that are configured in the thread configuration screens. These ports are used by external systems connecting to the system.

Ephemeral ports are allocated as sites/processes are run.

hcimonitor

This has a command port that is used by both the NetMonitor GUI and command line tools. It is used to communicate with the `hcimonitor` process to get statistics or control monitoring and alerting.

This is per-site.

There is one `hcimonitor` process for each site and each process has its command port.

The port number writes to `%HCISITEDIR%/exec/hcimonitor/cmd_port`.

hciengine

This process also has a command port that is used by command line tools. This port communicates with the `hciengine` process for controlling the message flow and threads.

This is per-process.

There can be many `hciengine` processes in each site and each one has its own command port.

The port number writes to `%HCISITEDIR%/exec/processes/process_name/cmd_port`.

Each protocol thread has an ICL port. This is per-thread.

This port is used for inter-process communication.

This can be made into a fixed port if the thread is configured as an inter-site destination.

Port range

On the **Site Preferences > System Management** tab, selecting **Enable Ephemeral Port Range** enables the **Port From** and **Port To** fields. The port range must have at least four ports. For example, specifying a range of 10000-10003 allocates ports 10000, 10001, 10002, and 10003.

- Valid **Port From** ports are 1025-65532.
- Valid **Port To** ports are 1028-65535.

The formula for determining the port range is:

Minimum port number = Total number of processes x 2 + total number of protocol threads + 1.

Explicit ports

Explicit ports should not belong to the port range.

If the port range is enabled, for example, a range of 10000~10050, and 10010 is set as an explicit ICL server port, then the 10010 port cannot be used simultaneously. If it is, then the thread which starts up later will not work.

Security Server Firewall tab

The **RMI Registry Port**, **RMI Object Port**, and **Exported Server Address** have the same validation rules as the host server.

When **RMI Registry Port** is cleared, the currently set value is retrieved and populated in the text box after the save. If the default RMI value of 14024 is set, then the `security.ini [firewall] rmi_registry_port` parameter is removed from the `security.ini` file.

When **RMI Object Port** is cleared, the text `Not Set` populates the text box. The `security.ini [firewall] security_server_default_port` parameter is removed from the `security.ini` file. Specifying a port in the text box fixes the port, so that the port does not change when the security server is restarted.

When **Exported Server Address** is cleared, the text `Not Set` populates the text box. The `security.ini [firewall] rmi_exported_server_port` parameter is removed from the `security.ini` file.

TCP/IP settings

TCP/IP settings are:

- **Enable RMI Registry for SSL**
This enables the RMI Registry for SSL on the Security Server to support Host Servers earlier than CIS 6.2. Those versions use SSL for the RMI connection. Otherwise, they cannot upgrade to the Security Server. By default, this setting is disabled.
- **RMI Registry Port for SSL**

This is one of two TCP/IP ports that are required for the CIS components with an SSL connection to communicate with one another. The default port is 14025. There is a similar Host Server setting that is required to match this one.

- **RMI Object Port for SSL**

This is the second of two TCP/IP ports that are required for the CIS components with an SSL connection to communicate with one another. By default, the Security Server randomly creates sockets on any available TCP/IP port. After this option is selected, the Security Server uses this TCP/IP port for all new socket connections.

Generating SSL certificates

Use these steps to generate SSL certificates:

- 1 Prepare a suite of certificates for the security server's SSL connection.
- 2 On the **Server Administration > Security Server tab > Firewall tab** select **Enable RMI Registry for SSL**, and then select **Generate SSL Certificates**.
- 3 This opens the **Generate SSL Certificates** dialog box. Add the **CA Certificate Location**, **Customer Name**, and **Password**.
- 4 Click **OK** to generate the certificate for the security server SSL connection. Ensure the CA and generated certificate meet the requirements.

Requirements are:

- Key size of CA cannot be greater than 1024; otherwise, an exception results.
- Key type is DSA. The size is 512 or 1024 (preferred).
- Signature algorithm is sha1WithDSA.

- 5 Place them in `security/certs/ssl`.

Security Server Advanced tab

If **JVM Arguments** is cleared, then the `server.ini [GENERAL] tab jvm_args` parameter is removed from the `server.ini` file.

Note: This parameter should be modified only after consulting Support.

When the host server is running under advanced security, a Security Server Options pane is on the **Advanced** tab. If the host server is running under no or basic security, then this pane does not show.

This pane has a field where you can set the **RMI Registry Port** that is used by security server.

By default, the host server uses the host server RMI registry port number to connect to security server. If any valid value is specified in this field, then the host server attempts to connect to the security server with that value.

The default value for this field is `not set` because usually it is not necessary to set this port.

If the host server tries to connect to different versions of security server, then the host server must change its RMI registry port number. This could affect all IDE client connectivity and might not be necessary.

Typically, the host server should use a different port number for security server connection purposes. This setting is set by specifying the port number for in the **Security Server RMI Registry Port** field.

This setting is especially important when multiple versions of the system are running on the same machine. The host server registry port number has to be different for these host servers to work correctly. They must use different port numbers to connect to a shared security server machine. You must configure this setting for each host server.

Security Server LDAP tab

The Cloverleaf security server has these functions:

- Imports users, groups, sub-groups, and hosts from one or more LDAP servers. Security server does not modify or create entries on LDAP.
- Downloads all ACLs from LDAP servers.
- Synchronizes ACLs automatically and manually through the GUI.
- Denies requests by users that are locked out or disabled on LDAP.
- Supports configuring ACLs against role or user.
- Logs user access events.

The security server **LDAP** tab contains all options to configure the LDAP server options for servers and user roles.

Synchronization section

This table shows the parameters for the synchronization section:

Parameter	Description
Mode	Incoming, Outgoing and Hybrid.
Cron	A time setting for triggering synchronizations.

LDAP Server section

This table shows the parameters in the LDAP Server section:

Parameter	Description
Host Name	The LDAP Server host name. Multiple LDAP servers should be separated by a comma.

Parameter	Description
Port	The LDAP Server port. If left empty, then the default port is used, depending on the selected Encryption Method .
Encryption Method	This is a list of options: No, SSL, StartTLS.
Authentication Method	This only supports Simple Authentication.
Default Domain Name	The default domain name that is used for the LDAP query.
Manager Distinguished Name	The Manager DN for connecting to LDAP servers.
Manager Password and Manager Confirm Password	The password for the Manager DN account.
Advanced	Opens the LDAP Advanced Configuration dialog box.
Test	Opens a testing dialog box for existing LDAP connection settings.

LDAP Advanced Configuration dialog box

This table shows the parameters for the **LDAP Advanced Configuration** dialog box:

Parameter	Description
Attribute for User ID	Specify an attribute name that is used for resolving the user ID from a list of LDAP attributes for a user. The default value is "sAMAccountName."
User Classes	Specify common classes for users. The default value is "organizationalPerson,person,top,user."
Group Classes	Specify common classes for groups. The default value is "group,top."
Server Classes	Specify common classes for servers. The default value is "computer,top."

Parameter	Description
User/Server/Group Search Base	<p>These fields are a distinguished name which specifies the LDAP entry under which all users/groups/servers are stored.</p> <p>For example, when the security server searches all users, it only searches entries under User Search Base.</p> <p>If this field is blank, then it uses a default value. The default value of all of the search bases is calculated from the default domain. For example, if the default domain is "infor.com," then for the user default base it would be "DC=infor, DC=com."</p>
ACL Organization Unit	<p>Specify an organization unit in the LDAP directory for storing ACL configuration for all resources in managed Cloverleaf host servers. During synchronization, the Cloverleaf security server synchronizes ACL configurations stored under ACL Organization Unit on the LDAP server and ACL configuration in the Cloverleaf security server.</p> <p>If this is not specified, then the Cloverleaf security server ignores the ACL synchronization. If the organization unit does not exist on the LDAP server, then the Cloverleaf Security Server creates it on the LDAP server. This is created when it first accesses the ACL organization unit on the LDAP server.</p>

LDAP information storage

LDAP Server configuration is stored in %HCIR00T\security\ssldap.xml.

Troubleshooting the LDAP configuration

If your testing result shows no query result, then:

- 1 Select **Advanced** to open the **LDAP Advanced Configuration** dialog box.
- 2 Remove the ou unit from the search bases in LDAP advanced configuration.
For example, in **User Search Base** change OU=Global-Users,DC=infor,DC=com to DC=infor,DC=com.
- 3 Do the same for **Group Search Base** and **Server Search Base**.
- 4 Test again.
- 5 If the testing result shows `invalidCredentials:49`, then on the **LDAP** tab check the **Encryption Method**, if it is SSL Encryption or StartTLS Extension.

6 Test again.

Security Server Encryption tab

Engine encryption provides a set of encryption/decryption APIs for all Cloverleaf applications.

Some administrator files, configuration files, and message data could contain sensitive data and can be encrypted. For example:

- Alert
- Database connection
- Global variable
- Lookup table
- NetConfig
- `hssecurity.ini`
- `security.ini`
- Error database
- SMAT database
- Raima

Support is provided for encryption of Cloverleaf configuration files and stored messages, through a default key and user-defined keys.

The engine encrypts data with the current public key. It decrypts data with the private key corresponding to the public key which is used for the original encryption.

The default public/private key is installed during installation. Users can override these default keys with user-specified keys.

User-specified public/private keys for administrative, configuration, and message data are configured using the **Encryption** tab.

Engine encryption validates the specified public/private key file. If validation is successful, then it checks if it is already registered in the SQLite database; if not, then it inserts an entry into the SQLite database.

Engine encryption keeps a history of public/private keys in a SQLite database, and can resolve the private key for the public key.

For migration, engine encryption supports importing keys from another SQLite database.

The server always encrypts the administration files and configuration files. The encryption of message data can be toggled on/off.

No attempt is made to encrypt or decrypt existing data. Existing objects are encrypted/unencrypted.

For example, when encryption of the SMAT database, error database, or internal Raima database is enabled, a password is required to protect the corresponding database. This password is encrypted.

Configure Encryption Keys panel

This section is where you generate and manage secure keys for both server administrators and users. Click **Generate** to generate the public and private key files. The key files are automatically registered to the SQLite database.

Engine encryption uses an asymmetric key encryption algorithm and supports two categories of public/private key pairs:

- A default key pair that is hardcoded and cannot be changed by users.
- Three types of user-specified key pairs that can be generated and registered with the **Server Administration** tool for administrative, configuration, and message data.

The engine encrypts data using the currently active public key of the specified type. The engine decrypts data with the private key corresponding to the public key that is used for the original encryption.

Engine encryption uses the OpenSSL RSA algorithm, and generates key files in PEM format.

Engine encryption keeps a history of key pairs in a SQLite database, so engine encryption can decrypt data that is encrypted with previously registered keys. The SQLite database is automatically by the system.

To support migration, engine encryption supports importing keys from another SQLite database.

This table shows the available options on the **Encryption** tab:

Option	Description
Configuration Files Key	<p>This key is used to encrypt sensitive information in Cloverleaf configuration files, including:</p> <ul style="list-style-type: none"> • All passwords (including, but not limited to, NetConfig protocols, alert emails, database connections, and web services security configurations). • Masked fields in lookup tables and global variables.
Administration Files Key	<p>This key is used to encrypt the security settings file and is reserved for use in encrypting administrative configurations and the audit log.</p> <p>The original <code>server.ini</code> is divided into <code>server.ini</code> and <code>hssecurity.ini</code>. All security server-related upgrades and the enable/disable audit log setting are moved to <code>hssecurity.ini</code>.</p> <ul style="list-style-type: none"> • <code>hssecurity.ini</code> is encrypted. • <code>server.ini</code> is not encrypted.

Option	Description
Messages Key	<p>This key is used to encrypt data, the SMAT database, and error and internal Raima databases.</p> <p>Data is encrypted with the current public key, and decrypted with the private key that corresponds with the public key used for the original encryption.</p> <p>Changes do not require a host server restart.</p> <p>The default key is installed with the software. If the configuration files, administration files, or messages keys are not specified, then the default key is used to encrypt or decrypt the sensitive data. It can be overridden with user-defined keys. For example, separate user-defined keys can be configured for data versus configuration.</p> <p>If an encryption key is not configured, then the default key is used for encryption.</p>

Use cases

The user requires the administration files, configuration file, and message data on two Cloverleaf host servers to be encrypted by the same user-specified keys. To do this, in the **Server Administration > Host Server > Encryption** tab:

- 1 On the first Cloverleaf host, click **Generate** for **Configure Files Key** and **Administration Files Key**.
- 2 On the second Cloverleaf host, click **Messages Key Import**, and specify the IP address or host name of the first Cloverleaf host in the **Import Encryption Keys** dialog box.
- 3 Select **Overwrite local configured keys** to import the encryption keys.

The user requires the administration files, configuration file, and message data on two Cloverleaf host servers to be encrypted by the same user-specified keys. These two host servers are in advanced security mode and are connected to the same security server. To do this:

- Open the **Server Administration** tool on the security server. In the **Security Server > Encryption** tab, click **Generate** for **Configure Files Key**, **Administration Files Key**, and **Messages Key**.
- Open the **Server Administration** tool on both host servers. On the **Host Server > Encryption** tab, click **Synchronize** to sync the encryption keys between the host server and security server.

The user requires to change the encryption keys. To do this, generate new encryption keys on the security server. Then, click **Synchronize** to sync the encryption keys between the host server and security server.

Security Server Web Server tab

The Security Server **Web Server** tab has these options:

- **Launch web server on Security Server startup**

When this is selected, the web server starts along with the security server. The default is selected.

- **Enable Cloverleaf API**

The Cloverleaf API uses a RESTful web service architecture and JavaScript Object Notation (JSON) as the data interchange format. Through this, you can create, edit, and delete Cloverleaf objects that were traditionally only available in the Cloverleaf GUI. The default is selected.

- **Enable Cloverleaf API Documentation**

Enable Cloverleaf API must be selected to enable this check box.

By default, **Enable Cloverleaf API Documentation** is cleared.

The CLAPI web page includes the CLAPI SDK document and examples. By default, it is disabled.

When the Cloverleaf security server is started with **Enable Cloverleaf API Documentation** enabled, all Cloverleaf APIs are listed in HTML through `https://hostservername:15149/clapi/`. This provides details about request parameters and responses for each API.

You can also enter API parameters and click **Try it out** in the API window.

The **Resources Available** page lists available APIs and their location.

For information on CLAPI, see [Cloverleaf API](#).

- Any changes enable **Save**. After you click **Save**, you must restart the security server before the change takes place.
- The HTTP port defaults to 15040, HTTPS defaults to 15043, and the Tomcat server shutdown port defaults to 15045. To change these ports, go to `$HCIR00T/security/tomcat/conf/server.xml`.

Security Server Database tab

The Security Server database has an events table that stores access requests for all hosts. As the number of access requests increases, the table size increases. This can consume much disk space.

To keep the table size manageable, you can specify the file limit size. Do this by configuring the number of days and frequency at which to purge the data. The Security Server then purges the events table on a regular basis according to the user's parameters.

For example, you can configure the table to delete all entries that exceed a specific number of days.

To configure the limit, select the **Automatically delete the records older than n day(s) every n hour(s)**. This feature is selected by default.

The default settings are:

- For number of hours, the default is 8.
- For number of days, the default is 7.

With the default settings, every 8 hours the system deletes all entries over 7 days of age.

This setting is saved in the `security.ini` file.

Note: When CLAPI is enabled and Security Server is started, Security Server calls CLAPI to enable all setting changes. When this operation is successful, it is unnecessary to restart Security Server. When CLAPI is disabled, or the previous operation has failed, you must restart Security Server to enable all setting updates.

Disk consumption management

This feature can resolve disk space issues for the Events table. Best practice is to monitor your disk consumption and adjust as necessary.

You can adjust the purge settings based on your real-time Events table data growth:

- If data increases rapidly, then use a shorter day/time interval for purging the data.
- If data increases too slow, then use a longer day/time interval for purging the data.

When this feature functions properly, the released disk space of the `..\integrator\security\data\cl_acls` folder returns to the operating system.

If disk consumption is not an issue, then you can disable this feature at **Server Administration > Security Server Database**.

System maintenance

The system should be the only application installed and running on your machine. Other applications running on the same machine as the system cause unnecessary performance problems. For example, disk space shortage, memory shortage, or the reduction of swap space:

- **Swap space:**
The engine shuts itself down if virtual memory on the machine is 90% exhausted. At 95% exhausted, the MonitorD does not start.
- **Full file system:**
All message saving is disabled when the `/integrator` file system is greater than 90% full. At 97% full, all engine output is disabled, and the engine shuts down. The MonitorD and engine do not start when the file system is 95% full or greater.

Part of maintenance is booting the system machine on a consistent basis. The system machine should be booted at least every two weeks. Rebooting helps prevent problems before they happen, and reduces calls to Support. Restarting your machine stops and removes zombie processes and resets virtual memory.

Auto-Start

Cloverleaf is a structured high availability scheme. Cloverleaf server clusters that are configured for high availability perform as a single integration system and provide almost 100% up time. This scheme is durable and operates continuously, providing protection from unplanned down times.

Auto-healing

Sometimes, failure can happen on the primary node (Cloverleaf machine). The auto-healing toolset assists Cloverleaf to transit runtime from a primary AS/Cluster node to a secondary AS/Cluster node. See [Cloverleaf auto-healing toolset](#). This toolset assists in:

- Host server and security server recovery.
- Cloverleaf engine and MonitorD recovery.
- Cloverleaf persistence layer recovery, consisting of recovery database, SMAT database, and so on.

Note: The MonitorD can be auto-started through the Global Monitor auto-start functionality. See **Configuring system and host level MonitorD auto-restart** in the Global Monitor online help.

Information storage

The Windows configuration is stored at %HCCIROOTDIR%\auto-start.

The Unix/Linux configuration is stored at \$HCCIROOT/auto-start.

On UNIX/Linux, auto-start scripts can be copied out of \$HCCIROOT during installation.

Auto-Start environment

The Auto-Start (AS) environment consists of these primary areas:

- Operating system/VM AS/Cluster environment, including:
 - Operating system: AIX, Linux, and Windows.
 - Cluster storage. For example, shared disk storage or SAN.
 - Network support, including IP pooling, virtual IP, private communication for heartbeats and cluster interconnect, and public network for user access.
 - AS monitoring and Resource Group Manager.
- Cluster interface to application
This contains information about the interface or configuration for application to run for AS action.
- Cloverleaf auto-healing toolset for:
 - Cloverleaf host server and security server recovery.
 - Cloverleaf engine and MonitorD recovery.
 - Cloverleaf persistence layer recovery, including the recovery database, SMAT database, and others.

Server interface

In Windows, Linux, and UNIX, clustering software is implemented to make Cloverleaf highly available. Two nodes (Cloverleaf machines) are configured: One is designated as the primary; the second is designated as the secondary.

Both nodes have access to a shared disk. The active node has the lock. If the primary fails, then the clustering software takes this action:

- The active node begins shutting down with these steps:
 - Stops the application by calling the AS stop script.
The stop action does not apply on a program failure.
 - Unmounts the shared disk at the Cloverleaf installed location.
 - Releases the Virtual IP (VIP) address. This is the Cloverleaf IP to external systems.
Note: This does not apply to a program failure stop action.
- The standby node (failing over):
 - Mounts the shared disk at the Cloverleaf installed location.
 - Assigns the VIP.
 - Starts the application by calling the AS start script.

VMWare Clustering on Linux and Windows

VMWare clustering, or virtual environment, monitors the hardware health. If the active server fails, then it relocates the VM's running on that node to its standby server. On the standby node, AS scripts are implemented as a service that starts Cloverleaf at boot time.

During implementation, a CloverAS service is created and configured as automatic at boot time. This service calls the AS start script.

Auto-Start Scripts options

These configuration items are available on the **Auto-Start Scripts** GUI:

Field/Option	Description
Sites	<p>These projects are controlled by auto-start scripts. When all projects are selected, they are always controlled by the auto-start scripts.</p> <p>It is unnecessary to update auto-start scripts for new projects.</p>
Excluded processes	<p>By default, all site processes are started by auto-scripts. Sites added to this list are excluded from auto-start.</p>
Sites repository file system	<p>When deploying CIS to the production environment, some sites that are created on replicated or shared file systems are linked to <i>HCIROOT</i>.</p> <p>When you link auto-start sites outside <i>HCIROOT</i>, you must set Sites repository file system.</p> <p>The auto-start scripts ensure that the link exists when the scripts start.</p>
Automatically start the host server on Windows	<p>This option applies in Windows systems. The existing NT service starts the host server. Setting this variable instructs the auto-start scripts to restart the host server.</p>

Field/Option	Description
Start sites by a single process	<p>In most cases, auto-start scripts create multiple processes to start a configured site, one process for one site. This causes a failover. Having many sites start up in parallel exhausts the system resources and causes unexpected errors.</p> <p>In these instances, select Defragment Database. When this is selected, sites are started one-by-one. This saves system resources, but runs slower.</p> <p>The single process mode saves logs on Windows at <i>HCIRoot/server/fake_stdout.txt</i> and <i>fake_stderr.txt</i>.</p> <p>On Linux, a site's startup log is created in sequence at <i>/var/log/message</i>.</p>
Defragment Database	<p>Running the <code>hcidbdefrag</code> command is helpful when you have a large number of messages in recovery .</p> <p>However, you must exercise caution. For example, if you have 200K messages, then the process takes many minutes, or hours, to complete. This option allows you to take control of the action, to avoid an unnecessarily long process time.</p>

Cloverleaf auto-healing toolset

The auto-healing toolset for the auto-start environment facilitates Cloverleaf in transitioning runtime when failure happens on the primary node. The transition is made from the primary auto-start/Cluster node to the secondary auto-start/Cluster node.

Auto-Start scripts

The auto-start scripts in this topic are what a cluster management system or an OS service management calls to start CIS runtime instances; for example, engine, monitor daemon, host server, and others.

The scripts can also recover the runtimes from the last failover. The scripts also clean up PID files, MSI shared files, and unlock the Lock Manager and Raima databases.

By default, if these files are not explicitly assigned a separate path during installation, they are installed under *HCIRoot/auto-start*. The installer has updated the scripts to point them to its installation. With this, you can copy the entire directory for use.

These files are under the auto-start directory:

- `as.hci.profile` is under the auto-start directory. This file contains all properties required to configure the auto-start logic. Every property is explained inside the file.
- Cloverleaf also looks for `auto-start.profile` under *HCIRoot/server* as an alternative to `as.hci.profile`.

You can copy `as.hci.profile` and rename it to take effect. Properties defined inside `HCIRoot/server/auto-start.profile` receive a higher priority, except for `AS_HCI_ROOT`, as the scripts use it to first find `HCIRoot`. The Cloverleaf installer updates it in `as.hci.profile` to point to its installation (without any additional configuration).

- `as.hci.start.pl` is under the auto-start directory. This is the entry point to invocation for the runtime startup. It is expected to be launched by the root user with the Perl installation inside CIS.

`as.hci.start.pl` usage:

```
as.hci.start.pl [-f profile] [-p] [-h] [-s] [-c]
```

Where:

- `-f profile` assigns the profile path for the start script. By default, the script attempts to locate `as.hci.profile` in the same directory.
- `-h` starts the host server.
- `-s` starts the security server.
- `-c` starts the scheduler.
- `-p` starts configured sites.

Notes:

By default, when not configured, `-h` and `-p` are "on". This script works as a compound autostart script for integration service.

When any one of `-h/-s/-p` is explicitly configured, the rest are "off" unless that one is also explicitly configured.

Service installation

Some HA systems reuse their OS service management when working for failover. The Cloverleaf installer on all supported OS platforms installs one or more CIS services. In this way, the CIS engine, monitor daemon, host server, and other runtime instances are brought up during OS startup.

- On Windows, a Windows NT Service is installed. Both the host server and auto-start script can be brought up by it after they are properly configured.
- On Linux and AIX, the CIS services can be installed by the `hciunixservice` script under `HCIRoot/sbin`. The script requires root privileges to fully work.

Usage:

```
hciunixservice -i h|s|e [auto-start script directory] [-c]
```

or

```
hciunixservice -r h|s|e
```

- `-i` installs the specified services. Additional auto-start scripts' install paths can be assigned, under which the Cloverleaf auto-start service is referred:
- `-r` removes the specified services.
- `h` is the host server service.
- `s` is the security server service.

- `e` is the cloverleaf scheduler service.
- `c` is a special option for CentOS 7 and later. This tunes `Systemd` unit files.

By default, assertions are made on `HCIRoot` for services and an error is reported to the system to mark the service as failure. Under cluster environment, `HCIRoot` is only available to the backup node when the major one is down. This option can be used to mute the alert. `Systemd` skips the service when `HCIRoot` is absent.

Note: The command must act as "root" to work.

System Services

For AIX and Linux distribution versions that have not introduced `Systemd` as the first process, the `hciunixservice` script installs the services under SysV style. Initial scripts are linked to `hostserver.service` or `securityserver.service` under `HCIRoot/sbin`, according to how Cloverleaf is installed.

Linux comes with `Systemd`.

On Linux, `hciunixservice` installs these services:

- Cloverleaf Integration Server:
 - `cloverleaf-integration.target`
 - `cloverleaf-autostart.service`
 - `cloverleaf-hostserver.service`
- Cloverleaf Security Server: `cloverleaf-securityserver.service`

For Cloverleaf Integration Server, `cloverleaf-integration.target` brings up the host server and auto-start services. These services bring up all expected monitor daemons and processes in configured sites.

The host server and auto-start are split into two services under `Systemd`. This utilizes the OS feature.

Host server is a stand-alone process in the system. A fork style service unit assists the system to ensure better control and to know the exact running status of the system.

Conversely, auto-start can create numerous processes. One service cannot monitor the status of all processes. The auto-start service is a "oneshot"-style service.

Cloverleaf Cluster

Clustering provides high availability by enabling application failover. The state of each individual node is periodically monitored and automatically relocates the application from a failed node to a designated secondary node.

When a failover happens, there might be a brief delay. In this case, refresh after the failover is finished.

Clustering can significantly reduce downtime and increase productivity by providing highly available service to all users.

Cluster has these key components:

- Hardware:
 - SAN (shared storage between cluster nodes)

- Private communication for heartbeats and cluster interconnect
- Public network for user access
- Local disks (unshared storage)
- Resource Group Manager
- CloverHA Agent
- Software:
 - OS (AIX, SUN OS, HP-UX, RHEL, and Windows)
 - Clustering Software (PowerHA, Sun Cluster, MC-Service Guard, Redhat Cluster, Windows Clustering and Veritas)
 - SAN Replication (IBM Metro Mirror, EMC Recover point and HP Storage works Multi-site Replication solution)
- SAN: Storage Area Network

This consists of disks that can be connected to more than one node at a time. Disks are made highly available through SAN. It can tolerate single node failure, since a physical path to the disks exists through the designated secondary node. It provides disk mirroring and stripping which protects against individual disk and node failures.
- Resource Group Manager

This provides mechanisms for high availability. This is policy-driven and automatically stops and starts the application on selected nodes. In the event of a node failure, it stops the application on a failed node and starts the application on the designated secondary node.
- CloverHA Agent

This is a dynamic agent with no user intervention. It is implemented as a shell script that gives maximum flexibility to users.

The Cloverleaf Cluster types are:

- Active-Passive
- Active-Test
- Active-Active
- Geo-Cluster

Active-Passive

In this configuration, node1 hosts Cloverleaf and node2 is designated as a standby node, waiting for node1 to fail.

When a failover happens, Cluster takes these actions:

- Moves the application data files to the secondary node.
- Moves the application IP address to the secondary node.
- Starts the application on the secondary node. If Clients notice a brief interruption in service, then they must refresh/reconnect.

Active-Test

In this configuration, node1 is hosting a production instance of Cloverleaf. Node2 is designated as the standby node and simultaneously is hosting the test instance.

When a failover happens, Cluster takes these actions:

- Moves the production data files to the secondary/test node.
- Moves the production IP address to the secondary/test node.
- Starts the production application on the secondary node. If Clients notice a brief interruption in service, then they must refresh/reconnect.

Active-Active

In this configuration, node1 is hosting the production instance of Cloverleaf. Node2 is hosting the second instance of Cloverleaf. Both nodes act as secondary to each other, also known as "mutual-take-over". When a failover happens, node2 becomes double-active.

Cluster takes these actions:

- Moves the production data files to the secondary node.
- Moves the production IP address to the secondary node.
- Clustering Software (PowerHA, Sun Cluster, MC-Service Guard, Redhat Cluster, Windows) Starts production application on the secondary node. If Clients notice a brief interruption in service, then they must refresh/reconnect.

Geo-Cluster

In this configuration, the cluster is configured across two data centers. This provides a greater level of availability and protection of storage by having a second copy of the data at the secondary location. When the primary data center fails, the application is relocated to the secondary/remote data center.

Cluster takes these action:

- Designates the remote data center as a primary location.
- Moves the application IP to the remote data center.
- Starts the application at the remote data center. If Clients notice a brief interruption in service, then they must refresh/reconnect.

Cluster scripts

Cluster scripts are located under `HCIR00T/auto-start`:

- `as.hci.profile`:
Property file to control sites.
- `as.hci.base.pl`:
Wrapper to set Cloverleaf environment.
- `as.hci.start.pl`:
Main startup script.
- `as.hci.stop.pl`:
Main shutdown script.
- `as.hci.siterun.pl`:

Startup of a site invoked for each site listed in the profile.

- `as.hci.sitestop.pl`:

Shutdown of a site invoked for each site listed in the profile.

Cluster event script environments

On Windows environments:

- 1 Create a "Generic" or "Generic Script" service to start/stop based on the cluster event. The sites are controlled by the `as.hci.profile` property file.

For additional information, see [Cloverleaf Cluster](#) on page 1163.

- 2 Edit `as.hci.profile` and enter a site list to be controlled (start/stop).

For example: `AS_HCI_SITES=site1,site2,site3`

On Linux environments:

- 1 Create a `systemd` resource to start/stop based on the cluster event.

The `systemd` scripts are located under `/usr/lib/systemd/system`.

Two `systemd` scripts are provided as part of the base install:

- To start Cloverleaf sites: `cloverleaf-autostart.service`
- To start the Cloverleaf Host Server: `cloverleaf-hostserver.service`

- 2 Edit `$HCIR00T/auto-start/as.hci.profile` and enter a site list to be controlled (start/stop).

For example: `AS_HCI_SITES=site1,site2,site3`

On UNIX-based systems:

- 1 Configure `as.hci.start.pl` for a start event.
- 2 Configure `as.hci.stop.pl` for a stop event.
- 3 Edit `$HCIR00T/auto-start/as.hci.profile` and enter a site list to be controlled (start/stop).

Failover support

For platform side support, template files for CentOS cluster settings are located inside the auto-start directory.

For the CIS built-in toolset, there are two parts to facilitate starting/recovering the CIS platforms/clusters in failover:

- Service support
- Auto-start script support

Service support

The CIS installer on all supported OS platforms can install the CIS services so that the CIS engine, MonitorD, Host Server, and other runtime instances can be brought up during OS startup.

- On Windows, a Windows Service is installed and can be managed the same way as other Windows services.
- On Linux and AIX, the CIS service is installed by the script `hciunixservice` under `HCIR00T`. For example, calling `hciunixservice -i h` as root installs the Host Server and the sites' auto-start services into the system.

For detailed usage, run the script directly.

Auto-start script support

For information, see [Cloverleaf auto-healing toolset](#) on page 1161.

Auto-Start deployment

For auto-start deployment, there are four basic types of OS/VM cluster environment and software configurations:

- **Active-Passive**
In this configuration, node1 is hosting Cloverleaf and node2 is designated as a standby node, waiting for node1 to fail. When a failover happens, Cluster takes this action:
 - Moves the application data files to the secondary node
 - Moves application IP address to the secondary node
 - Starts application on the secondary node; clients may see a brief interruption in service and may require a refresh/reconnect.
- **Active-Test**
In this configuration, node1 is hosting production instance of Cloverleaf. Node2 is designated as standby node, and hosts the test instance. When a failover happens, Cluster takes this action:
 - Moves the production data files to the secondary/test node.
 - Moves production IP address to the secondary/test node.
 - Starts production application on the secondary node; clients may see a brief interruption in service and may require a refresh/reconnect.
- **Active-Active**
In this configuration, node1 is hosting production instance of Cloverleaf. Node2 is hosting second instance of Cloverleaf. Both nodes act as secondary to each other, also known as mutual take-over. When a failover happens, node2 becomes double-active. Cluster takes this action:
 - Moves the production data files to the secondary node.
 - Moves production IP address to the secondary node.
 - Starts production application on the secondary node; clients may see a brief interruption in service and may require a refresh/reconnect.
- **Geo-Cluster**
In this configuration, the cluster is configured across two data centers. This provides a greater level of availability and protection of storage by having a second copy of the data at the secondary location. When a primary data center fails, application is relocated to secondary/remote data center. Cluster takes this action:
 - Remote data center is designated as a primary location.

- Moves the application IP to remote data center.
- Starts the application at remote data center; clients may see a brief interruption in service and may require a refresh/reconnect.

Auto-Start use cases

A System Engineer is tasked with server administration.

System reboot

The Administrator must reboot a Cloverleaf server. The Administrator shuts down all processes on the server and issues the reboot command. Upon start-up the auto-start scripts ensure the Cloverleaf Integration Service is running and starts all sites.

System failure

The clustering software has detected a network loss on the active server. The clustering software takes these actions:

- Calls a stop on the active server with the failure:
 - Stops the Cloverleaf application.
 - Unmounts the shared file system.
 - Releases the IP address.
- Calls a start on the standby/passive server:
 - Mounts the shared file system.
 - Assigns the IP address
 - Starts the Cloverleaf application and sites.

System backup

A backup for the system machine should be made at least once a week (for example, `mksysb`, `tar`, `cpio`). After completing the backup, verify the backed up information is valid by selecting and reading the media and data before storage.

Many backup software applications place locks on the files. This can cause corruption of the recovery database and the SMAT files. It can also panic an engine if an attempt is made to write to a log file that is locked.

In this case, if the `exec` folder is included in the backup:

- 1 Stop the site, including engines and daemons.
- 2 If the `exec` folder is backed up and restored, then messages previously delivered might be placed back into the recovery database. The result is out-of-context messages being delivered to downstream systems.

It is recommended not to restore the `exec` folder, or if you do, then reinitialize the recovery database before starting any engines.

Cycling files

Select **cycle output** on Network Monitor's **Process Controls** dialog box to cycle the `.log` files. This causes the current engine log (`.log`) and error (`.err`) files to close and be renamed with an `.old` extension. It then creates fresh `.log` files. The `.old` file is deleted. The `process.log`, and the `process.err` files can then be removed without harm.

Stop the process when manually moving or removing the `.log` or `.err` files. When you do this, a new one is created upon start-up of the process.

File system maintenance

Disk space maintenance is a priority for system administrators running the system engine. There are many log and saved message files, which should be consistently monitored and maintained to prevent disk space issues.

Log files which typically fill up are:

- `process.log (/integrator/sitename/exec/processes/process.log)`
- `process.err (/integrator/sitename/exec/processes/process.err)`
- SMAT files in the root directory

Other directories are:

- Database directory (`/integrator/sitename/exec/databases`)
- Error directory (`/integrator/sitename/exec/errors`)

Database directory maintenance

The databases (`/integrator/sitename/exec/databases`) directory can be problematic if not maintained. The system databases reuse disk space when messages are removed. These databases use disk space up to the maximum size of the file system. When a message is removed from a database, that file's disk space remains open to store a different message. When messages are added, the disk utilization is increased.

To clean up the database files:

Using `hcidbinit`

- 1 Run `showroot` to ensure that you are initializing the correct site.
- 2
- 3 Run `hcidbinit -A C` to initialize the databases and control files. Before initializing, the site daemons and all processes for the site in question must be down. In this way, when you initialize, it is site specific. This indicates that only the database files for that site are affected.
You can run `hcidbinit -e` or `hcidbinit -r`. Before initializing, the site daemons and all processes for the site in to reinitialize only that database and preserving the other one.

Using `hcidbdump`

- 1 Run `showroot` to ensure that you are initializing the correct site.
- 2 Run `hcidbdump {-r | -e} -D` to remove messages from the recovery and error database. It does not reclaim disk space, so this is not valid in this context.
- 3 Restart the engine.

Database file location

All information about the users, roles, and ACLs are stored in the embedded Derby database.

Derby is a full-featured, open source relational database management system (RDBMS) that is based on Java and SQL.

All database files are stored under `/integrator/sitename/security/data/cl_acls`.

UNIX system solutions

On a UNIX machine, there are several commands that help with disk space diagnosis. A few of these are:

- `df`
- `du`
- `compress`

For example:

- 1 Run `df` to get the file space percentages for each file system.
- 2 Use `du -sk *` at the beginning of the file system (usually `/integrator`).
- 3 Change to the largest directory.
- 4 Run the `du -sk *` command again.
Perform this routine until the largest files are found. After they are identified, analyze, move, remove, or compress them.

Windows system solutions

In Windows, line commands are not necessary. Determining the necessary information can usually be accomplished through a GUI.

For example:

- 1 Select a drive or a file.
- 2 Right-click and select **Properties**.
- 3 View properties, such as drive or file size and creation date.

Management tips

There are several tips that can help keep disk space usage at a minimum:

- Set all engine output levels to the lowest level possible, unless required for a specific troubleshooting instance.
- As the administrator, cycle the engine output and SMAT files often. You can do this by automating the cycle process. For example, cron job, perl script.
- Automate disk cleanup (See [Automating maintenance](#)).
- Cycle all log and saved message files on a regular basis.

Automating maintenance

Automate system maintenance using tools provided by the operating system. Automation is typically used within the system to:

- Cycle engine output
- Start and stop batch interfaces
- Reconfigure alerts
- Manage saved message files
- Schedule backups

Automating maintenance in UNIX

The UNIX utility `cron` is also called "the system alarm clock." It is a daemon that initiates other programs that are identified in the operating system's crontab, at an established time. The schedule is configured using the `crontab` command. Specifying and running a cron job is typically accomplished as root user.

To configure `crontab`:

```
minutes hours day-of-month month weekday username command
```

- *minutes* is minutes after the hour, with a range of 0-59.
- *hours* is the hour of the day, with a range of 0-23 (0=midnight).
- *day-of-month* is the day of the month, with a range of 1-31.
- *month* is the month of the year, with a range of 1-12.
- *weekday* is the day of the week, with a range of 0-6 (0=Sunday).
- *username* is the user name.
- *command* is the command or batch file.

To get a listing, use `crontab -l`.

To edit, use `crontab -e`.

For example, to set root and initiate a program:

```
05 0 * * 1 /usr/sbin/ksh -c 'eval `integrator/sbin/hcisetenv
    -root ksh /cis6.1/integrator/testsite`'
```

Automating maintenance in Windows

The Schedule Service is a Windows facility that schedules programs for delayed future execution and repeated periodic execution. Access to the Schedule Service is limited to the system administrator. By default, Schedule Service jobs run in the background.

Configuration:

```
at [\\host] time [when] command
AT [\\computername] [ [id] [/DELETE] | /DELETE [/YES]]
AT [\\computername] time [/INTERACTIVE]
[ /EVERY:date [,...] | /NEXT:date [,...]] command
```

- *host* is the name of the computer on which the task is to be scheduled.
- *time* specifies the time of day the command is to initiate.
- *command* is the command or batch program that is to run.
- *computername* specifies a remote computer. Commands are scheduled on the local computer if this parameter is omitted.
- *id* is an identification number that is assigned to a scheduled command.
- DELETE cancels a scheduled command. If *id* is omitted, then all scheduled commands on the computer are canceled.
- YES is used to cancel all job commands when no further confirmation is required.
- INTERACTIVE lets the job interact with the user's desktop, that is logged on at the time the job runs.
- EVERY:date[,...] runs the command on each specified days of the week or month.
If *date* is omitted, then the current day of the month is assumed.
- NEXT:date[,...] runs the specified command on the next instance of the day (for example, next Thursday).

If `date` is omitted, then the current day of the month is assumed.

Tcl leaks

Avoid the potential for memory leaks within custom Tcl code by always destroying `dat`, `grm`, and `msg` handles. How to build custom Tcl code is taught by Infor.

Use the TPS Testing Tool to test custom Tcl code for leaks.

Setting HTTP and SOAP headers in CAA-WS Clients

Users who have set up Clients in CAA-WS and must further tune clients for HTTP-level or SOAP-level headers can set up HTTP and SOAP headers in CAA-WS Clients.

There are some common HTTP headers that are configurable in http-conduits for Raw and SOAP/REST Clients.

This is accessible from WS Clients and Raw Clients.

For example, on the **WS Client** and **WS Raw Client** GUIs, when you select **Conduit(*.http-conduit)**, on the **Header** tab are several configuration fields for http-conduits.

On the **General** tab of the WS Raw-Client GUI, there is a Request Header Overrides table. On this table, Raw-Client consumers can set customized HTTP headers with no limit to the predefined headers in http-conduits. SOAP/REST consumers of WS-Clients have the same component.

SOAP extension configurations

SOAP extensions are appendixes to the standard SOAP specification. They are defined for a certain aspect of service functionalities.

Usually, the implementation of those extensions would be reflected in SOAP headers. It is irrelevant to support arbitrary SOAP headers in this aspect.

Options can be provided on the GUI with respect to the extension definition. Currently, options are provided that are related to WS-Addressing and WS-Security.

If you have an understanding of extension usage, then you can use those options to insert the expected SOAP headers without tuning the message.

WS-Addressing options are available on the SOAP consumer node and the bus node of the WS-Client. The configurable elements on the server are subject to the elements that are defined in <http://cxf.apache.org/schemas/ws-addr-conf.xsd>.

For WS-Security, configuration items are available on the **Policy Properties** tab of the **SoapConsumer(Soap12Withoutinput)** GUI. These include:

Panel	Configuration items
UsernameToken Properties	User name and password User name token validate.token enable.nonce.cache
X509 Certificate Handling Properties	Key password and file Keystore password and type
General Properties	callback-handler BSP compliance

Adding arbitrary HTTP headers in messages by overriding the userdata items

In addition to the GUI configuration, CAA-WS Clients and raw Clients also provide a series of userdata override APIs. These APIs provide additional flexibility on message customization for message-by-message in TPS.

You can add arbitrary HTTP headers to messages for both WS Clients and Raw Clients.

For example, to create a new TPS proc and set it for use on any UPOC point available on the message flow trace in the engine:

- This is found on the Network Configurator's **Outbound** tab at **TPS Outbound Data**. For example, Update ClientMessageAttachmentCont.
- In the TPS run mode code, the typical code piece to add HTTP headers is similar to:

```
keylget args MSGID mh
set userdata [msgmetaget $mh USERDATA]
keylset httpRequestHeaders <http header key> <the value for the key>
keylset userdata httpRequestHeaders $httpRequestHeaders
msgmetaset $mh USERDATA $userdata
```

- A complete table with available interfaces is located in the "Infor Cloverleaf Application Adapter Web Services User Guide" at **API > Client outbound overrides**.

For example, to add the Bearer type Authorization HTTP header in a request:

```
keylget args MSGID mh
set userdata [msgmetaget $mh USERDATA]
keylset httpRequestHeaders Authorization "Bearer $::token"
keylset userdata httpRequestHeaders $httpRequestHeaders
msgmetaset $mh USERDATA $userdata
```

- You can add any header with this method, regardless of whether it is public-defined.
- In the actual outbound HTTP message, this is found in the HTTP header section of your customizations.

Adding arbitrary SOAP headers for SOAP Consumers

Configurations are provided on the GUI that automatically add some SOAP extension headers for users. If you have a requirement to use private SOAP headers, then use an advanced approach to manipulate the SOAP headers. In this instance, have the SOAP consumer work in Message mode instead of Payload mode.

A Message mode Client requires more SOAP-specific knowledge than a Payload mode Client, in exchange for more flexibilities and control over messages.

For example, in this SOAP Envelope message, for a Payload mode Client you only provide the `query:AdhocQueryRequest/` message piece after the outbound TPS. This is three lines from the end of the example.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
    <Action xmlns="http://www.w3.org/2005/08/addressing">urn:ihe:iti:2007:RegistryStoredQueryResponse</Action>
    <MessageID xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:f198b7ab-ab39-41bd-91f3-d170ba38c1e7</MessageID>
    <To xmlns="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/addressing/anonymous</To>
    <RelatesTo xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:d5d34070-5d55-40f9-9763-1cb2a67fbd40</RelatesTo>
  </soap:Header>

  <soap:Body>
    <query:AdhocQueryRequest/>
  </soap:Body>
</soap:Envelope>
```

You only have direct control over the `query:AdhocQueryRequest/` section.

For a Message-mode Client, you provide the complete SOAP envelope after the outbound TPS. This involves much detail about SOAP messages. For example, you can mistake the SOAP envelope namespace for version SOAP version 1.1 and 1.2.

You have full control over the code within the `soap:Header` section.

Notes

You can prepare the messages at any point before the outbound TPS exits. You can also use any tool available in the engine, for example, translate, UPOC, and so on.

Clients ignore anything in private SOAP headers that are not publicly defined, even if you have also configured a public SOAP extension in the GUI. Configured public extensions, if there are conflicts with your custom SOAP headers, override user inputs. In this case, the SOAP extension is displayed on the GUI.

Configuration tips

Messages can be prepared at any point before the outbound TPS exits. You can use any available tool in the engine. For example, translate, UPOC, and so on.

Clients have no control over private SOAP headers that are not publicly defined, even when a public SOAP extension is configured in the GUI. For a configured public extension, any conflicts with a SOAP header that

is written by a user overrides user inputs. In this scenario, it is better to display the SOAP extension on the GUI.

For example:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
    <Action xmlns="http://www.w3.org/2005/08/addressing">urn:ihe:iti:2007:RegistryStored
QueryResponse</Action>
    <MessageID xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:f198b7ab-ab39-41bd-91f3-
d170ba38c1e7</MessageID>
    <To xmlns="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/addressing/anonymous</To>
    <RelatesTo xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:d5d34070-5d55-40f9-9763-
1cb2a67fbd40</RelatesTo>
  </soap:Header>
  <soap:Body>
    <query:AdhocQueryRequest/>
  </soap:Body>
</soap:Envelope>
```

This sample Tcl code piece can be used in TPS to manage SOAP headers:

```
set data [msgget $mh]
#using tdom to manipulate SOAP headers
package require tdom
set soapenvNs http://www.w3.org/2003/05/soap-envelope
set wsaNs http://www.w3.org/2005/08/addressing
set doc [dom parse $data]
set header [$doc getElementsByTagNameNS $soapenvNs Header]
set messageid [$doc createElementNS $wsaNs MessageID]
$messageid appendChild [$doc createTextNode "this is test message id"]
$messageid setAttributeNS $soapenvNs soapenv:mustUnderstand true
$header appendChild $messageid
msgset $mh [$doc asXML]
$doc delete
```

Setting up Security Server ACLs to enable GM functions

By default, only `application` under `integrator` is assigned with "administrator with run permission". All others have no administrator when CIS is upgraded to advanced mode.

You should also add the administrator user, with all permissions, to the site.

Note: The Notes and Site File Viewer GM widgets require the administrator permission to have "read" and "run" permission. On `site`, "write" and "run" permission is required in the ACL. This administrator permission setting also works on `none` admin users for the Notes and Site File Viewer widgets.

For example, `userA` is a non-admin user. To enable GM functions, having already given the administrator all site permissions, these permission settings must be configured to enable the related functions:

- MonitorD start/stop
 - **Application** > `hcinetmonitor userA` with run permission
 - **Command** > `hcsitectl userA` with run permission
- Process start/stop/restart
 - 1 Process Start
 - Application** > `hcinetmonitor userA` with run permission
 - Command** > `hciengineerun userA` with run permission
 - 2 Process Stop
 - Application** > `hcinetmonitor userA` with run permission
 - Command** > `hcienginestop userA` with run permission
 - 3 Process Restart
 - Application** > `hcinetmonitor userA` with run permission
 - Command** > `hciengineerestart userA` with run permission
- Thread-related commands
 - 1 Thread Start | Stop All Threads
 - Application** > `hcinetmonitor userA` with run permission
 - Command** > `hciengineerun userA` with run permission
 - Command** > `hccmd > pstart`
 - 2 Thread Stop | Start All Threads
 - Application** > `hcinetmonitor userA` with run permission
 - Command** > `hccmd > pstop` with run permission
 - 3 Thread Restart
 - Application** > `hcinetmonitor userA` with run permission

- Command > hciengineerun userA** with run permission
hcicmd > prestart > with run permission
- 4 Thread Hold
Application > hcinetmonitor userA with run permission
Command > hciengineerun userA with run permission
Command > hcicmd > phold_obd with run permission
 - 5 Thread Hold Reply
Application > hcinetmonitor userA with run permission
Command > hciengineerun userA with run permission
Command > hcicmd > phold_obd_reply with run permission
 - 6 Release Thread
Application > hcinetmonitor userA with run permission
Command > hciengineerun userA with run permission
Command > hcicmd > prls_obd with run permission
 - 7 Release Thread Reply
Site > Application > hcinetmonitor userA > with run permission
Site > Command > hciengineerun userA > with run permission
Site > Command > hcicmd > prls_obd_reply with run permission
Note: These operations also work on the List View.
- Configured Script
 To run site/process/thread-related commands with a configured script, you should also add this permission:
Site > Command > usercmd with run permission
 - SMAT
Site > Application > hcinetmonitor userA with run permission
Site > Command > hcicmd > resend_db with run permission for resend
Site > config > smat with read permission
 - ErrorDB
> integrator19.1 > Application > clapi with run permission
Site with read permission
Site > Application with run permission
Site > Command > hcicmd > resend_errordb with run permission

Cloverleaf security

In Transaction Control Protocol/Internet Protocol (TCP/IP), connections are established through sockets. Each socket is an identifier for a particular service on a particular network node. The socket consists of a node address that identifies the node, and a port number, which identifies the service.

Each TCP/IP connection is a single pipe between a client TCP/IP socket and a server TCP/IP socket. The client socket originates a connection request and the server socket receives it.

In normal operations, the client communicates through a client socket and the host server communicates through a server socket. In security administration operations, the host server communicates through a client socket and the security server communicates through a server socket.

Secure Sockets Layer (SSL) is a protocol that forms a separate layer on top of TCP/IP. SSL supports both encryption and authentication in internet communications.

Advanced security and basic security both use SSL to implement a secure connection for system configuration. Advanced security, which requires security server, gives complete and detailed control over access to the system message brokering system.

SSL functions

SSL functions include:

- Encryption: The data passed between programs is encrypted, preventing unauthorized persons from reading the data.
- Non-repudiation: Proves that the data came from a specific source.
- Authentication: Proves that the source is who they say they are.

Setting up TCP/IP SSL authentication

With mutual authentication, the Server and Client authenticate each other. This is common in web services.

To set up TLS in Cloverleaf (HTTP Client or TCP):

- 1 Use Portecle to create or manage a Java keystore and truststore.
 - The keystore contains the Server public/private key pair.

- The truststore contains the trusted Client's public certificates.
- 2** Use Portecle to create the PKCS12 keystore.
- 3** Use OpenSSL to export the Client certificate and the private key.
Curl uses the PEM format and cannot handle combined certificates. If necessary, then you can install OpenSSL.
- 4** Open the JKS keystore in Portecle.
- 5** Export the private key and head cert into a P12 file.
 - Export type: Private key and certificates
 - Export format: PKCS #12
- 6** Use OpenSSL to extract the individual certs.
- 7** Export the client public certificate into a PEM file. For example:

```
openssl pkcs12 in client_keystore.p12 out public.pem clcerts nokeys
```
- 8** Export the client private key into a PEM file. For example:

```
openssl pkcs12 in client_keystore.p12 out private.pem nocerts
```
- 9** Plug in the certificates created out of the Client keystore and the Server public certificate in the thread's SSL configuration.
 - If CN (Common Name) checking needs to be disabled, then add the cURL option:
CURLOPT_SSLVERIFYHOST 0
 - CA Path: the path where the cert files are
 - CA File: the remote hosts public cert
 - Certificate File: our public cert
 - Private Key: our private key
 - Password: the private key password

SSL types and modes

There are two types of SSL connections:

- Client SSL is for a client TCP/IP socket. Client SSL is the type for a client that is connected to a host server, and for a host server connected to a security server.
- Server SSL is for a server TCP/IP socket. Server SSL is the type for a host server that is connected to a client, and for a security server connected to a host server.

Every SSL connection must have a client SSL at one end and a server SSL at the other.

The mode determines whether messages sent by this end of the connection are to be encrypted. Then the mode determines if they are to be encrypted using a public key only or using both a public key and a private key.

If messages are to be encrypted using both keys, then the mode also determines whether messages received from the other end of the connection are to be authenticated.

SSL client modes

SSL client modes are:

- ClientAnon
- Client
- ClientAuth

ClientAnon

ClientAnon is the anonymous mode for clients. Clients use Ephemeral Diffie-Hellman, creating spontaneous public keys for encryption. The client is not authenticated, nor are private keys required. This is an appropriate choice for most secure web servers, which usually use anonymous SSL.

In Ephemeral Diffie-Hellman, the recipient randomly generates two numbers called the modulus and the generator, and transmits them to the sender. The sender uses those two numbers, along with the recipient's public key, to generate a temporary key. This key is for encrypting messages to be sent to that recipient only. The recipient independently generates the same key to decrypt the messages. When the connection is closed, both parties delete their temporary keys.

Features:

- Provides only encryption using ephemeral Diffie-Hellman, which applies a spontaneously generated public key to each message.
- No non-repudiation or authentication.
- No required certificates: these are dynamically generated when the connection is established.

Client

Client is the standard mode for clients. Clients use the private key and certificate for authentication. The client can be authenticated if requested, but it performs no authentication of the server. Use this when connecting to a web server, for it to authenticate you by your certificate.

Certificate File, **Private Key**, and **Password** are enabled.

You must specify the complete path for the certificate file and private key file. You must also provide a password, which is used to encrypt and decrypt the private key file.

Neither certificate file names nor private key file names can include spaces. The file extension for the certificate file or the private key file must be `.pem` for PEM-encrypted files.

This is the non-repudiation mode and is the standard mode for clients. The client does not perform any authentication of the server at the other end of the connection, but may be authenticated by the server.

Features:

- Provides encryption and non-repudiation. This indicates no authentication.
- Requires a private key and certificate. You specify the location of these files in the security dialog box.
- The private key is used to encrypt the data.
- The certificate is sent to the other party during handshaking.
- The other party uses the certificate to decrypt any data you send. This provides non-repudiation. This indicates the encrypted data could only have been generated from this private key.

ClientAuth

This is the authentication mode for clients. Clients use a private key and certificate for encryption, and a Certificate Authority to authenticate the server. Use this to authenticate the web server's certificate.

All text boxes are enabled.

You must specify the path for the CA file, CA file name, and complete path for the certificate file and private key file. You must also provide a password, which is used to encrypt and decrypt the private key file.

The CA file must reside on this computer system. This is the file that contains the CA for the other end of the connection whose path is required. If you are configuring SSL for a client, then you must provide directions to a local CA file for the corresponding server.

As you could have any number of connections, the CA files for various connections could be in various directories. CA Path should provide complete paths to all the directories to be searched for CA files. Each path is separated from the next by a semicolon.

Neither certificate file names nor private key file names can include spaces. The file extension for the certificate file or the private key file must be .pem for PEM-encrypted files.

Features:

- Messages are encrypted using a private key and certificate. The client obtains a certificate from the server at the other end and then validates that certificate using a CA file to authenticate the server. The client may also be authenticated by the server.
- Provides encryption, non-repudiation, and authentication.
- Requires a private key, certificate, and a CA certificate.

The CA certificate should be obtained from a trusted third-party.

The private key is used to encrypt the data.

The certificate is signed by both yourself and the CA certificate.

The certificate is sent to the other party during handshaking.

The other party verifies that your certificate is signed by a trusted CA.

The other party uses your certificate to decrypt the data.

The other party trusts that you are who you say you are. This is because your certificate has been signed by the CA and you both trust the CA. Optionally, one of you can be a CA and agree that you trust each other.

- The CA file is the path to the CA certificate, which is used for authenticating the other party's certificate.
- The CA Path is the directory where this and other CA files are located.

SSL server modes

SSL server modes are:

- ServerAnon
- Server
- ServerAuth

ServerAnon

This is the anonymous mode. It provides only encryption using ephemeral Diffie-Hellman, which applies a spontaneously generated public key to each message.

There are no required certificates, as these are dynamically generated when the connection is established.

No non-repudiation or authentication.

Server

This is the non-repudiation mode and is the standard mode for servers. The server does not perform any authentication of the client at the other end of the connection, but may be authenticated by the client.

It provides encryption and non-repudiation. This indicates no authentication.

It requires a private key and certificate. You specify the location of these files in the security dialog box.

- The private key is used to encrypt the data.
- The certificate is sent to the other party during handshaking.
- The other party uses the certificate to decrypt any data you send. This provides non-repudiation. The encrypted data could only have been generated from this private key.

ServerAuth

This is the authentication mode. Messages are encrypted using a private key and certificate. The server obtains a certificate from the client at the other end and then validates that certificate using a CA file to authenticate the client. The server may also be authenticated by the client.

It provides encryption, non-repudiation, and authentication.

It also requires a private key, certificate, and a CA certificate.

The CA certificate should be obtained from a trusted third-party.

- The private key is used to encrypt the data.
- The certificate is signed by both yourself and the CA.
- The certificate is sent to the other party during handshaking.
- The other party verifies that your certificate is signed by a trusted CA.
- The other party uses your certificate to decrypt the data.
- The other party trusts that you are who you say you are, because your certificate has been signed by the CA and you both trust the CA. Optionally, one of you can be a CA and agree that you trust each other.

The CA file is the path to the CA certificate, which is used for authenticating the other party's certificate.

The CA path is the directory where this and other CA files are located.

The server modes closely resemble the client modes. The modes are also symmetrical across the connection; that is, the client mode selected for one end of the connection must be balanced with the server mode selected for the other end.

Mode matches

This table shows the available mode matches for various selected modes:

Mode	Available match
ClientAnon	The server mode must be: ServerAnon.
Client	The server mode must be: Server or ServerAuth.
ClientAuth	The server mode must be: ServerAuth or Server.
ServerAnon	The client mode must be: ClientAnon.
Server	The Client mode must be: Client or ClientAuth.
ServerAuth	The client mode must be: ClientAuth or Client.

OpenSSL ciphers

The supported OpenSSL ciphers are:

ECDHE-RSA-AES256-GCM-SHA384	TLSv1.2	Kx=ECDH	Au=RSA	Enc=AES-GCM(256)	Mac=AEAD
ECDHE-RSA-AES256-GCM-SHA384	TLSv1.2	Kx=ECDH	Au=RSA	Enc=AES-GCM(256)	Mac=AEAD
ECDHE-RSA-AES256-SHA384	TLSv1.2	Kx=ECDH	Au=RSA	Enc=AES(256)	Mac=SHA384
ECDHE-RSA-AES256-SHA	SSLv3	Kx=ECDH	Au=RSA	Enc=AES(256)	Mac=SHA1
ECDHE-ECDSA-AES256-SHA	SSLv3	Kx=ECDH	Au=ECDSA	Enc=AES(256)	Mac=SHA1
SRP-DSS-AES-256-CBC-SHA	SSLv3	Kx=SRP	Au=DSS	Enc=AES(256)	Mac=SHA1
SRP-RSA-AES-256-CBC-SHA	SSLv3	Kx=SRP	Au=RSA	Enc=AES(256)	Mac=SHA1
SRP-AES-256-CBC-SHA	SSLv3	Kx=SRP	Au=SRP	Enc=AES(256)	Mac=SHA1
DH-DSS-AES256-GCM-SHA384	TLSv1.2	Kx=DH/DSS	Au=DH	Enc=AES-GCM(256)	Mac=AEAD
DHE-DSS-AES256-GCM-SHA384	TLSv1.2	Kx=DH	Au=DSS	Enc=AES-GCM(256)	Mac=AEAD
DH-RSA-AES256-GCM-SHA384	TLSv1.2	Kx=DH/RSA	Au=DH	Enc=AES-GCM(256)	Mac=AEAD

DHE-RSA-AES256-GCM-SHA384	TLSv1.2	Kx=DH	Au=RSA	Enc=AES-GCM(256)	Mac=AEAD
DHE-RSA-AES256-SHA256	TLSv1.2	Kx=DH	Au=RSA	Enc=AES(256)	Mac=SHA256
DHE-DSS-AES256-SHA256	TLSv1.2	Kx=DH	Au=DSS	Enc=AES(256)	Mac=SHA256
DH-RSA-AES256-SHA256	TLSv1.2	Kx=DH/RSA	Au=DH	Enc=AES(256)	Mac=SHA256
DH-DSS-AES256-SHA256	TLSv1.2	Kx=DH/DSS	Au=DH	Enc=AES(256)	Mac=SHA256
DHE-RSA-AES256-SHA	SSLv3	Kx=DH	Au=RSA	Enc=AES(256)	Mac=SHA1
DHE-DSS-AES256-SHA	SSLv3	Kx=DH	Au=DSS	Enc=AES(256)	Mac=SHA1
DH-RSA-AES256-SHA	SSLv3	Kx=DH/RSA	Au=DH	Enc=AES(256)	Mac=SHA1
DH-DSS-AES256-SHA	SSLv3	Kx=DH/DSS	Au=DH	Enc=AES(256)	Mac=SHA1
DHE-RSA-CAMELLIA256-SHA	SSLv3	Kx=DH	Au=RSA	Enc=Camellia(256)	Mac=SHA1
DHE-DSS-CAMELLIA256-SHA	SSLv3	Kx=DH	Au=DSS	Enc=Camellia(256)	Mac=SHA1
DH-RSA-CAMELLIA256-SHA	SSLv3	Kx=DH/RSA	Au=DH	Enc=Camellia(256)	Mac=SHA1
DH-DSS-CAMELLIA256-SHA	SSLv3	Kx=DH/DSS	Au=DH	Enc=Camellia(256)	Mac=SHA1
ECDH-RSA-AES256-GCM-SHA384	TLSv1.2	Kx=ECDH/RSA	Au=ECDH	Enc=AES-GCM(256)	Mac=AEAD
ECDH-ECDSA-AES256-GCM-SHA384	TLSv1.2	Kx=ECDH/ECDSA	Au=ECDH	Enc=AES-GCM(256)	Mac=AEAD
ECDH-RSA-AES256-SHA384	TLSv1.2	Kx=ECDH/RSA	Au=ECDH	Enc=AES(256)	Mac=SHA384
ECDH-ECDSA-AES256-SHA384	TLSv1.2	Kx=ECDH/ECDSA	Au=ECDH	Enc=AES(256)	Mac=SHA384
ECDH-RSA-AES256-SHA	SSLv3	Kx=ECDH/RSA	Au=ECDH	Enc=AES(256)	Mac=SHA1
ECDH-ECDSA-AES256-SHA	SSLv3	Kx=ECDH/ECDSA	Au=ECDH	Enc=AES(256)	Mac=SHA1
AES256-SHA256	TLSv1.2	Kx=RSA	Au=RSA	Enc=AES(256)	Mac=SHA256
AES256-SHA	SSLv3	Kx=RSA	Au=RSA	Enc=AES(256)	Mac=SHA1
CAMELLIA256-SHA	SSLv3	Kx=RSA	Au=RSA	Enc=Camellia(256)	Mac=SHA1
PSK-AES256-CBC-SHA	SSLv3	Kx=PSK	Au=PSK	Enc=AES(256)	Mac=SHA1

ECDHE-RSA-AES128-GCM-SHA256	TLSv1.2	Kx=ECDH	Au=RSA	Enc=AES-GCM(128)	Mac=AEAD
ECDHE-ECDSA-AES128-GCM-SHA256	TLSv1.2	Kx=ECDH	Au=ECDSA	Enc=AES-GCM(128)	Mac=AEAD
ECDHE-RSA-AES128-SHA256	TLSv1.2	Kx=ECDH	Au=RSA	Enc=AES(128)	Mac=SHA256
ECDHE-ECDSA-AES128-SHA256	TLSv1.2	Kx=ECDH	Au=ECDSA	Enc=AES(128)	Mac=SHA256
ECDHE-RSA-AES128-SHA	SSLv3	Kx=ECDH	Au=RSA	Enc=AES(128)	Mac=SHA1
ECDHE-ECDSA-AES128-SHA	SSLv3	Kx=ECDH	Au=ECDSA	Enc=AES(128)	Mac=SHA1
SRP-DSS-AES-128-CBC-SHA	SSLv3	Kx=SRP	Au=DSS	Enc=AES(128)	Mac=SHA1
SRP-RSA-AES-128-CBC-SHA	SSLv3	Kx=SRP	Au=RSA	Enc=AES(128)	Mac=SHA1
SRP-AES-128-CBC-SHA	SSLv3	Kx=SRP	Au=SRP	Enc=AES(128)	Mac=SHA1
DH-DSS-AES128-GCM-SHA256	TLSv1.2	Kx=DH/DSS	Au=DH	Enc=AES-GCM(128)	Mac=AEAD
DHE-DSS-AES128-GCM-SHA256	TLSv1.2	Kx=DH	Au=DSS	Enc=AES-GCM(128)	Mac=AEAD
DH-RSA-AES128-GCM-SHA256	TLSv1.2	Kx=DH/RSA	Au=DH	Enc=AES-GCM(128)	Mac=AEAD
DHE-RSA-AES128-GCM-SHA256	TLSv1.2	Kx=DH	Au=RSA	Enc=AES-GCM(128)	Mac=AEAD
DHE-RSA-AES128-SHA256	TLSv1.2	Kx=DH	Au=RSA	Enc=AES(128)	Mac=SHA256
DHE-DSS-AES128-SHA256	TLSv1.2	Kx=DH	Au=DSS	Enc=AES(128)	Mac=SHA256
DH-RSA-AES128-SHA256	TLSv1.2	Kx=DH/RSA	Au=DH	Enc=AES(128)	Mac=SHA256
DH-DSS-AES128-SHA256	TLSv1.2	Kx=DH/DSS	Au=DH	Enc=AES(128)	Mac=SHA256
DHE-RSA-AES128-SHA	SSLv3	Kx=DH	Au=RSA	Enc=AES(128)	Mac=SHA1
DHE-DSS-AES128-SHA	SSLv3	Kx=DH	Au=DSS	Enc=AES(128)	Mac=SHA1
DH-RSA-AES128-SHA	SSLv3	Kx=DH/RSA	Au=DH	Enc=AES(128)	Mac=SHA1
DH-DSS-AES128-SHA	SSLv3	Kx=DH/DSS	Au=DH	Enc=AES(128)	Mac=SHA1
DHE-RSA-SEED-SHA	SSLv3	Kx=DH	Au=RSA	Enc=SEED(128)	Mac=SHA1
DHE-DSS-SEED-SHA	SSLv3	Kx=DH	Au=DSS	Enc=SEED(128)	Mac=SHA1
DH-RSA-SEED-SHA	SSLv3	Kx=DH/RSA	Au=DH	Enc=SEED(128)	Mac=SHA1

DH-DSS-SEED-SHA	SSLv3	Kx=DH/DSS	Au=DH	Enc=SEED(128)	Mac=SHA1
DHE-RSA-CAMELLIA128-SHA	SSLv3	Kx=DH	Au=RSA	Enc=Camellia(128)	Mac=SHA1
DHE-DSS-CAMELLIA128-SHA	SSLv3	Kx=DH	Au=DSS	Enc=Camellia(128)	Mac=SHA1
DH-RSA-CAMELLIA128-SHA	SSLv3	Kx=DH/RSA	Au=DH	Enc=Camellia(128)	Mac=SHA1
DH-DSS-CAMELLIA128-SHA	SSLv3	Kx=DH/DSS	Au=DH	Enc=Camellia(128)	Mac=SHA1
ECDH-RSA-AES128-GCM-SHA256	TLSv1.2	Kx=ECDH/RSA	Au=ECDH	Enc=AES-GCM(128)	Mac=AEAD
ECDH-ECDSA-AES128-GCM-SHA256	TLSv1.2	Kx=ECDH/ECDSA	Au=ECDH	Enc=AES-GCM(128)	Mac=AEAD
ECDH-RSA-AES128-SHA256	TLSv1.2	Kx=ECDH/RSA	Au=ECDH	Enc=AES(128)	Mac=SHA256
ECDH-ECDSA-AES128-SHA256	TLSv1.2	Kx=ECDH/ECDSA	Au=ECDH	Enc=AES(128)	Mac=SHA256
ECDH-RSA-AES128-SHA	SSLv3	Kx=ECDH/RSA	Au=ECDH	Enc=AES(128)	Mac=SHA1
ECDH-ECDSA-AES128-SHA	SSLv3	Kx=ECDH/ECDSA	Au=ECDH	Enc=AES(128)	Mac=SHA1
AES128-GCM-SHA256	TLSv1.2	Kx=RSA	Au=RSA	Enc=AES-GCM(128)	Mac=AEAD
AES128-SHA256	TLSv1.2	Kx=RSA	Au=RSA	Enc=AES(128)	Mac=SHA256
AES128-SHA	SSLv3	Kx=RSA	Au=RSA	Enc=AES(128)	Mac=SHA1
SEED-SHA	SSLv3	Kx=RSA	Au=RSA	Enc=SEED(128)	Mac=SHA1
CAMELLIA128-SHA	SSLv3	Kx=RSA	Au=RSA	Enc=Camellia(128)	Mac=SHA1
PSK-AES128-CBC-SHA	SSLv3	Kx=PSK	Au=PSK	Enc=AES(128)	Mac=SHA1
ECDHE-RSA-RC4-SHA	SSLv3	Kx=ECDH	Au=RSA	Enc=RC4(128)	Mac=SHA1
ECDHE-ECDSA-RC4-SHA	SSLv3	Kx=ECDH	Au=ECDH	Enc=RC4(128)	Mac=SHA1
ECDH-RSA-RC4-SHA	SSLv3	Kx=ECDH/RSA	Au=ECDH	Enc=RC4(128)	Mac=SHA1
ECDH-ECDSA-RC4-SHA	SSLv3	Kx=ECDH/ECDSA	Au=ECDH	Enc=RC4(128)	Mac=SHA1
RC4-SHA	SSLv3	Kx=RSA	Au=RSA	Enc=RC4(128)	Mac=SHA1

RC4-MD5	SSLv3	Kx=RSA	Au=RSA	Enc=RC4(128)	Mac=MD5
PSK-RC4-SHA	SSLv3	Kx=PSK	Au=PSK	Enc=RC4(128)	Mac=SHA1
ECDHE-RSA-DES-CBC3-SHA	SSLv3	Kx=ECDH	Au=RSA	Enc=3DES(168)	Mac=SHA1
ECDHE-ECDSA-DES-CBC3-SHA	SSLv3	Kx=ECDH	Au=ECD-S SA	Enc=3DES(168)	Mac=SHA1
SRP-DSS-3DES-EDE-CBC-SHA	SSLv3	Kx=SRP	Au=DSS	Enc=3DES(168)	Mac=SHA1
SRP-RSA-3DES-EDE-CBC-SHA	SSLv3	Kx=SRP	Au=RSA	Enc=3DES(168)	Mac=SHA1
SRP-3DES-EDE-CBC-SHA	SSLv3	Kx=SRP	Au=SRP	Enc=3DES(168)	Mac=SHA1
EDH-RSA-DES-CBC3-SHA	SSLv3	Kx=DH	Au=RSA	Enc=3DES(168)	Mac=SHA1
EDH-DSS-DES-CBC3-SHA	SSLv3	Kx=DH	Au=DSS	Enc=3DES(168)	Mac=SHA1
DH-RSA-DES-CBC3-SHA	SSLv3	Kx=DH/RSA	Au=DH	Enc=3DES(168)	Mac=SHA1
DH-DSS-DES-CBC3-SHA	SSLv3	Kx=DH/DSS	Au=DH	Enc=3DES(168)	Mac=SHA1
ECDH-RSA-DES-CBC3-SHA	SSLv3	Kx=ECDH/RSA	Au=ECDH	Enc=3DES(168)	Mac=SHA1
ECDH-ECDSA-DES-CBC3-SHA	SSLv3	Kx=ECDH/ECD-S SA	Au=ECDH	Enc=3DES(168)	Mac=SHA1
DES-CBC3-SHA	SSLv3	Kx=RSA	Au=RSA	Enc=3DES(168)	Mac=SHA1
PSK-3DES-EDE-CBC-SHA	SSLv3	Kx=PSK	Au=PSK	Enc=3DES(168)	Mac=SHA1
EDH-RSA-DES-CBC-SHA	SSLv3	Kx=DH	Au=RSA	Enc=DES(56)	Mac=SHA1
EDH-DSS-DES-CBC-SHA	SSLv3	Kx=DH	Au=DSS	Enc=DES(56)	Mac=SHA1
DH-RSA-DES-CBC-SHA	SSLv3	Kx=DH/RSA	Au=DH	Enc=DES(56)	Mac=SHA1
DH-DSS-DES-CBC-SHA	SSLv3	Kx=DH/DSS	Au=DH	Enc=DES(56)	Mac=SHA1
DES-CBC-SHA	SSLv3	Kx=RSA	Au=RSA	Enc=DES(56)	Mac=SHA1

Java cipher suites

A list of the available Java suites is also available.

See <https://docs.oracle.com/javase/8/docs/technotes/guides/security/SunProviders.html#SunEC>.

Cloverleaf default password strength rules

These password rules are used by default:

- It must have 8-15 characters.
- It must use upper and lowercase characters.
- It must have at least one digit.
- It must not contain the user name.

If necessary, then you can edit the password rule to add additional rules. These are not valid by default. For example:

- Three (3) or more identical characters can be forbidden. For example, `aaa` or `111`.
- Three (3) or more consecutive characters can be forbidden. For example, `abc` or `123`.

To keep malicious users from guessing the user password, CIS records the count of the number of failed log-in attempts for a user account. The account is locked if the log in fails more than five (5) times.

By default, the lock is released automatically after 15 minutes. This gives users the opportunity to log in again.

Definitions

This table shows common security features:

Feature	Definition
Private key	This is a file that contains a digital key which is used to encrypt data and sign certificates. For additional security, this file can be encrypted with a password. You should keep this file in a secure area on your machine and never give it out.
Public key	This is generated based on your private key. This is a digital key that others can use to decrypt data that you have encrypted with your private key.
Certificate	This is a file containing a public key, plus information on who the certificate belongs to (name, address, and so on). The certificate is digitally signed using a private key, and can be signed by more than one party. Certificates are generally signed by yourself and a trusted third party, such as Verisign.
CA certificate	This is a "certificate authority" certificate that contains only a single public key. This is usually generated by a trusted third party.

Self-signed certificates

This topic describes how to generate a self-signed root CA, issue inherited certificates, and use them in CIS.

Self-signed certificates are generated using OpenSSL 1.1.1.

Cloverleaf only supports the RSA key type.

Preconditions are:

- Ensure the `rootca-openssl.cnf` and `Customer-openssl.cnf` configuration files are under the current `OpenSSL` directory. By default, this is `OpenSSL/bin`.
- Ensure there are no remaining files from the previous certs generation under the current `OpenSSL` directory. By default, this is `OpenSSL/bin`.
- Generate a random file at first. To do this:

```
OpenSSL> rand -base64 -out .rnd 100
```

root CA configuration file

This is a list the root CA configuration file contents. These parameters are save in the `rootca-openssl.cn` file.

- `RANDFILE = .rnd`
- `string_mask = pkix`
- `[req]`
 - `default_bits = 1024`
 - `default_keyfile = keyfile.pem`
 - `distinguished_name = req_distinguished_name`
 - `prompt = no`
 - `output_password = mypass`
 - `req_extensions = v3_req`
- `[req_distinguished_name]`
 - `CN = rootca`
 - `C = US`
 - `L = Atlanta`
 - `ST = GA`
 - `O = ROOTCA`
 - `OU = Customer Support`
- `[v3_req]`
 - `basicConstraints = critical, CA:true`
 - `keyUsage = critical, keyCertSign, cRLSign`
 - `nsCertType = client, sslCA`

customer CA configuration file

This is a list the customer CA configuration file contents. These parameters are save in the `Customer-openssl.cnf` file.

- RANDFILE = .rnd
- string_mask = pkix
- [req]
 - default_bits = 1024
 - default_keyfile = keyfile.pem
 - distinguished_name = req_distinguished_name
 - prompt = no
 - output_password = mypass
 - req_extensions = v3_req
- [req_distinguished_name]
 - CN = Customer
 - C = US
 - L = Columbus
 - ST = OH
 - O = Customer
 - OU = Department
- [v3_req]
 - basicConstraints = critical, CA:true
 - keyUsage = critical, keyCertSign, cRLSign
 - nsCertType = client, sslCA

Generating an RSA key self-signed root CA

- 1 Generate a key pair for the root CA, whose key size is 2048:

```
OpenSSL> genrsa -out rootca-keypair.pem 2048
```

- 2 Generate a self-signed root CA cert with the rootca-openssl.cnf config file. The signature digest is sha512:

```
OpenSSL> req -config rootca-openssl.cnf -new -x509 -key rootca-keypair.pem  
-out rootca-cert.pem -extensions v3_req -days 3650 -sha512
```

- 3 Convert the cert format from PEM to DER:

```
OpenSSL> x509 -inform PEM -in rootca-cert.pem -outform DER -out rootca-cert.der
```

- 4 Encrypt the root CA private key to DER. Set the password to 111112Aa.

```
OpenSSL> pkcs8 -topk8 -in rootca-keypair.pem -inform PEM -outform DER -out  
enc-rootca-key.der -passout pass:111112Aa -v1 PBES2 -v2prf hmacWithSHA512 -iter 4096
```

Generating an RSA key customer certificate

To generate a customer certificate request and sign it with the root CA:

- 1 Generate a key pair for the customer CA with the key size 2048:

```
OpenSSL> genrsa -out Customer-keypair.pem 2048
```

- 2 Generate the Customer cert request with the Customer-openssl.cnf config file:

```
OpenSSL> req -config Customer-openssl.cnf -key Customer-keypair.pem  
-new -out Customer-req.pem
```

- 3 Sign the certificate request with the root CA. The signature digest is sha512

```
OpenSSL> x509 -req -in Customer-req.pem -out Customer-cert.pem -extfile Customer-openssl.cnf  
-extensions v3_req -CA rootca-cert.pem -CAkey rootca-keypair.pem -CAcreateserial -days 365  
-sha512
```

- 4 Convert the Customer cert format from PEM to DER:

```
OpenSSL> x509 -inform PEM -in Customer-cert.pem -outform DER -out Customer-cert.der
```

- 5 Encrypt the Customer private key and convert its format from PEM to DER:

```
OpenSSL> pkcs8 -topk8 -inform PEM -in Customer-keypair.pem -outform DER -out enc-Customer-  
key.der  
-passout pass:111111Aa -v1 PBES2 -v2prf hmacWithSHA512 -iter 4096
```

Installing the Host Server and Security Server

- 1 Install the Host Server by installing Cloverleaf Integration Services.
- 2 Copy these certs into the server/certs folder:
 - rootca-cert.der
 - Customer-cert.der
 - enc-Customer-key.der
- 3 Install the Security Server.
- 4 Copy rootca-cert.der into the security/certs folder.
- 5 Ensure you have the Security Server license. Then, you can start it.

Upgrading to basic and advanced security

Upgrade the Host Server to basic security with the customer CA cert and restart.

Upgrade the Host Server to advanced security using *hostname* as the security server host. Then, restart.

Launching the IDE

- 1** Create users. Open the IDE. In the User Login dialog box, create a new user.
For example, `userA`.
- 2** Approve users.
Open the Certificate Manager. Log in with `Customer/111111Aa`. Then, approve `userA`.
- 3** Grant permissions.
Open the ACL/Role Manager dialog box to modify the user ACLs of `userA`.
- 4** Log in to the IDE using `userA`.

Generating certificates

- 1** Create a certificate request, which has been signed with your private key.
- 2** Give the certificate request to the CA.
- 3** The CA signs the certificate request with their own private key.
This generates a certificate that has been signed by both you and the CA.

Algorithms

The RSA algorithm is used to create private the key.

Note: SSL is not supported in CIS versions later than 6.2. Instead, TLS is used.

Authentication

When SSL supports authentication and encryption, the certificate and private key must be authenticated by a trusted third party called a CA (Certificate Authority).

Infor is the root (ultimate) CA for the system. Infor certifies the authenticity of the security administrator for customer organizations. Any Infor-certified administrator can in turn act as a CA for other members of the customer organization.

SSL authentication is accomplished by examining a CA file. This file contains the name and public key of the certified user and is digitally signed by the CA. If there is a chain of certification, then each CA file in the chain must contain information that identifies it as a descendant of the root.

To authenticate the server, the client must have the server's CA file. Similarly, to authenticate the client, the server must have the client's CA file.

Command line interaction with CIS security

Most command lines are designed as local commands and have no interaction with the Security Server. The best practice recommendation is to use the related CLAPI, CLWizard function, or GUI tool.

The command line `hcisiteinit` is designed as a local command line to create a site that is installed on the Host Server.

For example, the `hcisiteinit` command line only creates a site on the Host Server under Basic/Advanced Security mode. The Site Init GUI tool, CLAPI, or CLWizard have log-in features for user authentication (Basic/Advanced Security).

In Advanced Security, they also check the user permissions to create a site and upload the created site to Security Server.

The Site Init GUI is a client-side tool that interacts with the Host Server and calls `hcisiteinit` to create a site on the Host Server.

The Site Init GUI performs these tasks under Basic/Advanced Security mode:

- User log-ins to the selected site. It sets up the TLS connection to the Host Server and checks whether the user has run permission on the Site Init tool.
- Requests the Host Server to create a site with `hcisiteinit` and verifying if the user has permission on `hcisiteinit` running.
- Updates `server.ini`.
- Connects with the Security Server to upload the created site.
- Audits who and when logs in to the Site Init GUI.

siteSecurityInfo file

In earlier CIS versions, any user could turn on/off SMAT/error/internal database encryption in the IDE's **Site Preferences** dialog box. This defeats HIPAA/security control when you are hosting in the Cloud.

In CIS, only the administrator can turn on/off encryption in the ST Cloud.

These options are located on the **Server Administration** GUI. On this GUI, the administrator can choose to switch to SMAT, error, or internal database encryption for a site. The corresponding site configuration information is saved in the `site/siteInfo` file.

There is no option for users to switch between SMAT, error, or internal database encryption on the **Site Preferences** dialog box.

Disk encryption, which cannot be switched to "on" or "off" by users, satisfies the HIPAA compliance regulations. SMAT database encryption is another level of encrypted data that adds additional security beyond the HIPAA requirements.

For Cloverleaf Cloud, SMAT database encryption is restricted and enforced at the admin level. This is not a user option. This applies to the error and internal databases.

With this level of security, for users who create a site, the SMAT/error/internal databases are saved into an encrypted database. There is no option for users to choose not to save into the encrypted database.

siteSecurityInfo

In earlier CIS versions, the internal, error, and SMAT database encryption settings were stored in `siteInfo`. For security reasons, these settings are now stored in the `$HCISITEDIR/siteSecurityInfo` file.

The database encryption settings are configurable only on the **Server Administration** dialog box.

For `hciengine` and `hcimonitor`, if there is no `siteSecurityInfo` file in the current site, or a required file item is missing, then the default values are used to access the databases. In this case, each database is processed as encrypted, using the default password.

For the database-related tools, if there is no `siteSecurityInfo` file in the current site, the tools exit with errors. If the required items are not found, then the default values are used.

Key names in the `siteSecurityInfo` file are case insensitive.

The database-related tools include:

- `hcidbinit`
- `hcidbencrypt`
- `hcidbcheck`
- `hcidbdefrag`
- `hcidbconvert`
- `hcidbdump`
- `hcidbsetvers`

Affected files are:

- `bin/hciengine(.exe)`
- `bin/hcimonitor(.exe)`
- `clgui/lib/DBSearchInterface.jar`
- `bin/hcidbcheck(.bat)`
- `bin/hcidbconvert(.bat)`
- `bin/hcidbencrypt(.bat)`
- `bin/hcidbdefrag(.bat)`
- `bin/hcidbdump(.exe)`

- bin/hcidbinit(.bat)
- bin/hcidbsetvers(.exe)
- bin/hcismatcycle(.bat)
- bin/hcismatconvert(.bat)
- bin/hcismatcrypt(.bat)
- bin/hcisiteinit(.bat)
- bin/hcicreatesite(.bat)
- bin/hcctl(.exe)
- bin/hcirootcopy(.bat)

Database encryption: \$HCISITEDIR/siteSecurityInfo

These keys for the raima/error/SMAT databases are saved into `siteSecurityInfo`:

- `dbencryption=`
- `internaldbkey`
- `errordbencryption=`
`errordbkey=`
- `smatencrypted=`
`smatdriver=`
`smatdbkey=`

Decryption

Cloverleaf internal database configuration:

- `dbencryption` is the encryption/decryption flag for the internal database. This includes the recovery database, and others.
 - 0: Plain text database files.
 - 1: Encrypted database files.
- `internaldbkey` is the internal database encryption key.
If `dbencryption` is set and `internaldbkey` is blank, then the default encryption key is used.

Cloverleaf error database configuration:

- `errordbencryption` is the encryption/decryption flag for the sqlite error database.
 - 0: Plain text database files.
 - 1: Encrypted database files.
- `errordbkey` is the error database encryption key.
If `errordbencryption` is set and `errordbkey` is blank, then the default encryption key is used.

SMAT database configuration:

- `smatencrypted` determines if the SMAT database is encrypted. The default is encrypted.
 - 0 indicates not encrypted.

- 1 indicates encrypted.
 - `smatdriver` is the SMAT driver type. The default is `sqlite`.
 - `file` indicates the SMAT file in which SMAT messages are stored.
 - `sqlite` indicates that SMAT messages are stored in the SMAT database.
 - `smatdbkey` is the SMAT database encryption key. By default, this is not used if blank.
- Note:** `smatencrypted=1` and `smatdriver=file` are an invalid combination.

Database encryption: `hcicreatesite`

```
hcicreatesite [-c site] [-p path]
[-E SMATDBFlag] [-D type] [-K SMATDBKey]
[-I InternalDBFlag] [-M InternalDBKey] [-R ErrDBFlag]
[-N ErrDBKey] root newsite
```

- `-c site` is the site to be cloned by `newsite`.
 - If `-c site` is used and no flags are used, then database encryption stays consistent with the configuration in the template site.
 - If `-c site` is not used and no flags are used, then the databases are encrypted by default and use the default key.
- `-p path` is the target path where `newsite` is saved.

A symbolic link `$HCIR00T/newsite` is created linking to `<path>/newsite` after site creation. This is available only in NTFS on Windows. The target path must be R/W to the user.
- `-E SMATDBFlag` encrypts/decrypts SMAT.
 - 1 indicates encrypted. This is the default.
 - 0 indicates unencrypted.
- `-D type` is the driver type that is used to read/write SMAT. Acceptable values are `file` and `sqlite`. When `-D type` is `sqlite`, the `sqlite` database retains the same behavior as the internal and error databases.
- `-K SMATDBKey` is the encryption key of the SMAT database.
- `-I InternalDBFlag` is the internal database. This flag has a higher priority than the encryption setting in the template site specified by `-c site`.
 - 1 indicates encrypted. This is the default.
 - 0 indicates unencrypted.
- `-M InternalDBKey` is the encryption key of the internal database.
- `-R ErrDBFlag` is the error database. This flag has a higher priority than the encryption setting in the template site specified by `-c site`.
 - 1 indicates encrypted. This is the default.
 - 0 indicates unencrypted.
- `-N ErrDBKey` is the encryption key of the error database.
- `root` is the name of the root to create the new site under `newsite`.
- `newsite` is the name of the new site to create. This must not currently exist.

Note: The listed parameters pertain to database encryption. Additional parameters are available for this command. See [hcicreatesite](#).

Database encryption: hcisiteinit

```
hcisiteinit [-c site] [-s] [-i] [-p path] [-E SMATDBFlag]
[-D type] [-K SMATDBKey] [-I InternalDBFlag] [-M InternalDBKey] [-R ErrDBFlag]
[-N ErrDBKey] newsite
```

- `-c site` is the site to be cloned by `newsite`.
 - If `-c site` is used and no flags are used, then database encryption stays consistent with the configuration in the template site.
 - If `-c site` is not used and no flags are used, then the databases are encrypted by default and use the default key.
- `-s` runs `hcisitectl -s` after site creation.
- `-i` ignores restrictions on the site name.
- `-p path` is the target path where `newsite` is saved.
A symbolic link `$HCIRoot/newsite` is created linking to `<path>/newsite` after site creation. This is available only in NTFS on Windows. The target path must be R/W to the user.
- `-E SMATDBFlag` encrypts/decrypts SMAT.
 - `1` indicates encrypted. This is the default.
 - `0` indicates unencrypted.
- `-D type` is the driver type that is used to read/write SMAT. Acceptable values are `file` and `sqlite`. When `-D type` is `sqlite`, the `sqlite` database retains the same behavior as the internal and error databases.
- `-K SMATDBKey` is the encryption key of the SMAT database.
- `-I InternalDBFlag` is the internal database. This flag has a higher priority than the encryption setting in the template site specified by `-c site`.
 - `1` indicates encrypted. This is the default.
 - `0` indicates unencrypted.
- `-M InternalDBKey` is the encryption key of the internal database.
- `-R ErrDBFlag` is the error database. This flag has a higher priority than the encryption setting in the template site specified by `-c site`.
 - `1` indicates encrypted. This is the default.
 - `0` indicates unencrypted.
- `-N ErrDBKey` is the encryption key of the error database.
- `newsite` is the name of the new site to create. This must not currently exist.

Database encryption: hcirootcopy

```
hcirootcopy [-f] [-n] [-v] [-N] [-s sitelist] [-l listfile]
[-p path] [-e encoding] [-u username] [-w password]
[-E SMATDBFlag,InternalDBFlag,ErrDBFlag] [-D type] [-r] [-c] sourceRoot
```

- `-f` specifies no query.
- `-n` specifies test query.
- `-v` specifies verbose mode.
- `-N` specifies to not copy the database.

- `-s sitelist` copies a single site or colon-separated list of sites. For example, `site1` or `site1:site2:site3`
- `-l listfile` copies the sites that are listed in *listfile*.

listfile should be newline formatted. For example:

```
site1
site2
site3
```

- `-p path` is the actual path where *sitelist* is to be saved.
A `$HCIR00T/site` symbolic link under the current root is created as a blank placeholder file to the actual path of site path/site after site creation.
This option is supported only on UNIX.
- `-e encoding` is the encoding of the source files. If this option is omitted, then the default value is `utf-8`.
- `-u username` is your organization's end-user username.
- `-w password` is your organization's end-user password.
- `-E SMATDBFlag,InternalDBFlag,ErrDBFlag`

These specify whether the SMAT, internal, or error database is encrypted.

SMATDBFlag is used for the SMAT database.

InternalDBFlag is used for the Raima database.

ErrDBFlag is for error database:

- `1` indicates encrypted.
- `0` indicates unencrypted.
- `-1` indicates no change. This is the default. If no flag is specified, then it keeps the same configuration as the previous site.

For example, `-E -1,1,1` indicates that SMAT encryption has no change after migration, and the internal/error databases are encrypted after migration.

- `-D type` is the driver type that is used to read/write SMAT. Acceptable values are `file` or `sqlite`.
If `-D type` is `sqlite`, then the `sqlite` database retains the same behavior as the internal and error databases.
- `-r` copies the `server/server.ini` file.
Note: This option ignores the security server-related keys.
- `-c` updates the encryption method.
- `sourceRoot` specifies the source root directory.

Database encryption: hcidbencrypt

The error database is in a SQLite database, where you can edit, resend, and search the error database. A user interface similar to SMAT is available to manage the database.

This command is provided for users to change the error and internal database passwords.

```
hcidbencrypt {-e|-I} [-d] [-p password] [-q]
```

Both the error and internal database are supported for encrypting/decrypting with a user-defined password.

- `-e` specifies encryption/decryption of the error database.
- `-I` specifies encryption/decryption of the internal databases. For example, the recovery database..
- `-d` specifies to do decryption, when it is given; otherwise, encryption.
- `-p password` specifies the encryption key.
If this option is not specified, then the default encryption key is used.
- `-q` indicates quiet, or no output except errors.

To change the error database password:

- Decrypt the error database or internal database using the old password:

```
hcidbencrypt {-e|-I} [-d] [-p] oldpassword [-q]
```

- Encrypt the error database or internal database with the new password:

```
hcidbencrypt {-e|-I} [-d] [-p] newpassword [-q]
```

If the `-p` option is not used in the command, then the engine uses the default password to encrypt/decrypt the database.

siteSecurityInfo keys

These keys in `siteSecurityInfo` are updated based on the encryption/decryption option:

- `dbencryption`
The default is `1`: Raima databases are encrypted.
- `internaldbkey`
The default is `""`: Use the default Raima database key for encryption.
- `errordbencryption`
The default is `1`: Error database is encrypted.
- `errordbkey`
The default is `""`: Use the default error database key for encryption.

`siteSecurityInfo` is encrypted. The keys that are saved in this file are plain text.

Encryption

SSL supports these encryption algorithms:

- Ephemeral Diffie-Hellman: This type is used for anonymous SSL, which does not require a certificate or a private key.
- Triple-DES: This type is used for user name/password SSL, which requires both a certificate and a private key. Triple-DES is so-called because it repeats DES (Defense Encryption Standard) encryption three times. This effectively doubles the length of the original DES key from 56 bits to 112 bits.

TLS is used for providing communications security over a computer network, and requires both a certificate and a private key. The certificate and the private key are in two separate files. These are encrypted using algorithms that conform to a standard called X.509. The signature algorithm of the public key is SHA512withRSA.

The algorithm that is used to encrypt the private key is PBEwithHmacSHA512AndAES_256.

Encryption methods

The files that contain certificates and private keys must use one of two encryption methods:

- **DER (Distinguished Encoding Rules)**
Sometimes known as ASN.1, this uses binary data. A DER file can be transmitted as an attachment to an email message. This is the preferred method.
- **PEM (Privacy Enhanced Mail)**
This uses printable ASCII data. A PEM file is an ASCII version of a DER file, typically used when an email recipient cannot accept attachments. A PEM file can be cut and pasted into an email message.

It is not necessary for both files to use the same method. For example, your organization might generate its own DER private key files, but obtain PEM certificate files from elsewhere.

Protocol GUI security settings

Access the GUI security functions through the HTTP Client, TCP/IP, or PDL TCP/IP protocol.

For example, to access the security settings through the HTTP Client protocol, select the **HTTPS** check box on the **HTTP Client Protocol Properties** dialog box.

Then, click **Configure**. This opens the **HTTPS** dialog box, where security settings are configured.

Mode options for HTTP Client, PDL TCP/IP are:

- **ClientAnon**
- **Client**
- **ClientAuth**

For TCP/IP, select **Server** and click **SSL** and then **Configure**. Mode options are:

- **ServerAnon**
- **Server**
- **ServerAuth**

The remainder of the security information depends on the selected mode.

SSL protocol

In the **SSL** dialog box, select the openSSL version from the **SSL Protocol** list.

When you select an SSL protocol, a description of the selected protocol is shown in the comment field together with the Mode description.

- When **TLSv1.3** is selected, this message is shown in the comment field:

A client will send out TLSv1.3 client hello messages and will indicate that it only understands TLSv1.3.

A server will only understand TLSv1.3 client hello messages.

- When **All** is selected, this message is shown in the comment field:

A client will send out TLSv1.3 client hello messages and will indicate that it only understands TLSv1.3.

A server will only understand TLSv1.3 client hello messages. This is the suggested method for server configuration.

- All** is the default for Client and Server.

TLSv1.3 supports these cipher suites:

- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256
- TLS_AES_128_GCM_SHA256
- TLS_AES_128_CCM_SHA256
- TLS_AES_128_CCM_8_SHA256

Note: TLSv1.3 does not support anonymous mode.

CA Path and CA File

CA Path is the Certificate Authority directory path.

CA File is the Certificate Authority file name.

In the **SSL** dialog box, **CA Path** and **CA File** are available only when **ClientAuth** or **ServerAuth** is selected as the mode.

CA File is the certificate used for authentication.

- If **CA File** is specified, then it points to a file containing CA certificates in PEM format. This file can contain more than one certificate. Certificates in the file are surrounded by "-----BEGIN CERTIFICATE REQUEST-----" and "-----END CERTIFICATE REQUEST-----". Other text can happen between certificates, for example, a description of the certificate.
- If **CA Path** is specified, then it points to a directory containing CA certificate files in PEM format. Each file in **CA Path** can contain only one certificate. The name of each file in **CA Path** must be the hash value of the subject name. If two subject names have the same hash value, then different extensions can be used (that is, <hash_value>.1 and <hash_value>.2).

CA File or **CA Path** can be used to specify the sources of CA certificates.

- **Certificate File**, the certificate file name, is used for authentication and encryption. This option is available only when Client/Server or ClientAuth/ServerAuth is selected for **Mode**.
- **Private Key**, the private key file name, is used to create digital signatures. This option is available only when Client/Server or ClientAuth/ServerAuth is selected for **Mode**.
- **Password** is used if your private key is password-encrypted. The driver uses this password to decrypt the private key before using it. This option is available only when Client/Server or ClientAuth/ServerAuth is selected for **Mode**.

Examples of client/server negotiation

When the system is used as a client when **TLSv1** is selected:

- If the remote server has TLSv1 support, then the handshake result is TLSv1.
- If the remote server does not have TLS, then an error is returned and the connection attempt fails. Handshake failure messages are reported to the engine log.

When the system is used as a client, and **All** is selected, if the remote server has TLSv1 support, the handshake result is TLSv1. Otherwise, depending on priorities, the server may choose SSLv2 or SSLv3 as the result.

Standard/version selection priorities are determined by the server; TLSv1 has the highest priority. This operation succeeds unless the server does not support any of the requested standards/versions.

When the system is used as a server and **SSLv3** is selected and the remote client places an initial connection request with SSLv3 or All, the handshake result is SSLv3. All other types get a failure and an error returned.

The system is used as a server when **All** is selected.

Any client-selected standard is used because **All** means "accept all standards." If the client side is also **All**, then the default standard is used. Default depends on the client side application. For example, openssl sets SSLv2 as the default. An application using GnuTLS sets SSLv3 as the default.

Upgrading and downgrading security

To give your organization the greatest possible flexibility, the system is installed with no security. Your organization can add these security features at any time.

Security features can be added only on the computer that runs the host server. To use advanced security, host server and client must be connected to a security server on a different computer.

To upgrade security, use the Security Upgrade utility.

- If you currently run the engine with no security, then you can upgrade to basic security or advanced security. Upgrading to advanced security automatically adds basic security.
- If you currently run the engine with basic security, then you can upgrade to advanced security.

Advanced security can be added only on a computer system that runs host server. To enable security administration, you must add advanced security to one and only one host server, no matter how many are

included in your system. If your system already has basic security, then you must add advanced security on the same computer system where basic security is located.

The `SecurityUpgrade.log` file logs information from the `SecurityOptions` class. This class contains the upgrade information. For example, "none to basic," "none to advanced," "basic to advanced," and so on.

The `SecurityUpgrade.log` file is generated in the `HCIR00T` folder.

Notes

- In a system with multiple host servers, security features should be added to each host server.
- Certificates must be issued before any security upgrade.

Running CIS with basic or advanced security

In basic security, the IDE and host server establish a TLS 1.2 connection.

In advanced security, the IDE, host server, and security server establish a TLS 1.2 connection.

This connection uses `TLS_DHE_DSS_WITH_AES_256_CBC_SHA256`.

Upgrading/Downgrading from the command line

A `mode` parameter is available on the command line for different combinations of security upgrade/downgrade.

Security levels

Security levels are:

- **No security:** This determines access to resources by the network's security. All network traffic is in clear text.
A system with no security must have at least one engine and a host server for every engine, and may have any number of clients.
- **Basic security:** This enables the security administrator of your organization's system to issue certificates that identify individuals who are authorized to access the system, and revoke those certificates. In a system with basic security, only individuals who have valid certificates can log in to the system.
The major component of basic security is the Certificate Manager, which the security administrator uses to issue, modify, and revoke certificates.
- **Advanced security:** This provides fine-grained control over access to the message brokering system. With advanced security, the security administrator can set user-specific permissions. This defines not only which resources each user can access, but also which operations each user can perform on each resource. Advanced security also enables the creation of roles to which multiple users can belong, and set role-level permissions that apply to all role members.
This level requires security server installation.
Note: Advanced security is a premium feature that is sold separately as an add-on. Contact your Account Manager for additional information.

Security passwords

During the security upgrade procedure, you are prompted for several passwords that are required to link security features with your system. You are also prompted to authorize access to the host server and to the security management tools.

Any and all necessary passwords are created by your security administrator. Make certain that these passwords and other items, such as port numbers, host names/numbers, and location of certificates are safely recorded for future reference. These must be kept in a secure location.

The main advantage of having the administrator issue passwords instead of Infor is that the private key and passwords are never exposed. Request and certificate files, when transmitted, only contain public information.

This ensures that only your organization has access to any private information and passwords.

This table shows the different password levels:

Password level	Description
Customer password	This password is associated with the unique customer name that is specified for your organization by Infor. This is required to access the system and issue user certificates. The issuer is Infor.
Host server password	This password is required to access certificate files that are maintained on the host server. The issuer is the customer.
Administrator certificate password	This password is required to access the administrator certificate. The issuer is the customer.

Note: You should record passwords that are assigned by your organization and keep them in a secure location.

Before upgrading security

Before upgrading security, you must request a public certificate file and a private key file. Create a case on the Infor Support Portal or Concierge to request generation of these files. Provide the name in which the certs are to be generated and copy the files to your computer.

These are the directory locations of the certificate files:

- `enc-customer-key.der`, where *customer* is the unique customer name that is assigned to your system by Infor.
- `customer-cert.der`, where *customer* is the unique customer name that is assigned to your system by Infor.

You must also verify that these files do not exist:

- `$HCIR00T/cert/truststore_clserver.jks`
- `$HCIR00T/cert/keystore_clserver.jks`

If they do, then you must delete both files before beginning the upgrade process.

If this is not accomplished, then the existing files cause the upgrade process to fail.

After this, you can begin the upgrade.

- 1 If you have installed the host server and client, then restart your computer, or restart Windows.
- 2 Stop all threads, the Monitor Daemon (`monitord`), and the engine. If necessary, then use the Windows Task Manager.
- 3 Exit and log out all GUIs.
- 4 Obtain the necessary user IDs and passwords for the security level to which you plan to upgrade.

Upgrading CIS 6.2.x host server to advanced security using CIS 19.1+ security server

The CIS 6.2.x host server and CIS 19.1+ security server can use different CA certificate algorithms of since the algorithm has been upgraded on CIS 19.1+. When this happens, the host server cannot directly upgrade to advanced security mode.

The host server and security server must import each others CA to its own truststore. This indicates that they trust each other. Then, a TLS connection can be built for a successful security upgrade.

To import the CA:

- 1 Verify that the truststore is available on the host server and security server.
On the host server, the truststore is named `truststore_clserver.jks`. This is located at `%HCIR00T%/server/certs`. If this file is not available, then upgrade the host server to basic mode. Then, the file is automatically generated.
On the security server, the truststore is named `truststore_clsecurity.jks`. This is located at `%HCIR00T%/security/certs`.
- 2 Manually import the CA of CIS 6.2.x host server to `truststore_clsecurity.jks` of the CIS 19.1+ security server.
- 3 Restart the CIS 19.1+ security server.
- 4 Manually import the CA of CIS 19.1+ security server to `truststore_clserver.jks` of the CIS 6.2.x host server.
- 5 Upgrade the CIS 6.2.x host server to advanced mode using the CIS 19.1+ security server.
Note: The SHA256withDSA algorithm is not supported by JRE1.8.0_60. This is used in CIS 6.2.3 and previous versions. If the DSA algorithm has been used by the CA on the CIS 19.1+ security server, then upgrade the host server to CIS 6.2.3 or later version.

Upgrading from none to basic security

If your system currently has no security, then use this procedure to upgrade to basic security.

- 1 At the command prompt for your operating system, change to the directory where you installed the system, for example, `C:\cloverleaf`.

- 2** Specify `hciupgradeutility`.

This starts the security upgrade utility and opens the first dialog box, which lists the sites in this system and shows the current security level.

You can also launch the security upgrade utility from **Start > All Programs > Infor Cloverleaf Integration Suite > Tools > Security Upgrade**.
- 3** Click **Continue with Upgrade**.

The utility continues to the next step where you select the security level.
- 4** Select **Basic Security** and click **Next**.
- 5** Specify whether to install the Certificate Manager.

Select **Yes** to install the security management tools (Certificate Manager). If the only host server is the one on this computer, then the Certificate Manager must be installed. If you have multiple host servers, then install the Certificate Manager on only one computer. If you already have host server certificates, for example, from a previous installation, then Setup opens a dialog box. This dialog box prompts whether to use the existing certificates or to generate new ones.

Select **No** to omit the installation of the Certificate Manager. If multiple host servers are on your system, then click **No** for every host server except one.
- 6** Specify the location of the certificate files.

The upgrade utility asks for the directory that contains the two essential certificate files that your organization received from Support. See Step 2 of [Before upgrading security](#). Specify the certificate files location path in the field. You can also click **Browse** and select the correct directory from the file browser.

If you have not obtained these two files and copied them to your computer, then you cannot complete the basic security Setup process. Click **No** to cancel the setup operation, and exit the Setup utility.
- 7** If you have specified the directory, then click **Next**, and specify this information:

In **Customer Name**, specify the unique customer name that is assigned to your organization.

In **Password**, specify the customer password given to you by Infor.
- 8** Click **Next**.
- 9** Specify the password assigned by your organization to authorize access to certificate files that are maintained on the host server. If you do not know this password, then contact your security administrator. Click **Next**.
- 10** Specify the password to use to authorize access to the administrator certificate. This certificate is created automatically by the security upgrade utility. The security administrator for your organization can now log in to create certificate files for other users. Click **Next**.
- 11** The security upgrade utility opens a message dialog box that notifies you that the security change is complete. Click **OK**.
- 12** Restart your computer or restart Windows to complete the upgrade.

Upgrading from none to advanced security

- 1** At the command prompt for your operating system, change to the directory where you installed the system, for example, `C:\cloverleaf`.
- 2** Specify `hciupgradeutility`.

This starts the security upgrade utility and opens the first dialog box, which lists the sites in this system and shows the current security level. You can also launch the security upgrade utility from **Start > All Programs > Infor Cloverleaf Integration Suite > Tools > Security Upgrade**.

- 3 Click **Continue with Upgrade**.
- 4 Select **Advanced Security**, and click **Next**.
- 5 Specify whether to install Certificate Manager. If the only host server is the one on this computer, then you must install the Certificate Manager on this computer. If you have multiple host servers, then install the Certificate Manager only on one computer. If you already have host server certificates, then Setup opens a dialog box asking whether to use the existing certificates or to generate new ones.
Select **Yes** to install the security management tools (Certificate Manager).
Select **No** to omit the installation of the Certificate Manager. If you have multiple host servers, then you should click **No** for every host server except one.
- 6 Specify the complete path where the certificate files are located, or browse to the correct directory.
If you have not obtained these two files and copied them to your computer, then you cannot complete the basic security setup process. Click **No** to cancel the setup operation, and exit the Setup utility.
- 7 If you specified the directory, then click **Next**.
In **Customer Name**, specify the unique customer name that is assigned to your organization.
In **Password**, specify the customer password given to you by Infor, then click **Next**.
- 8 Specify the password assigned by your organization to authorize access to certificate files that are maintained on the host server. If you do not know this password, then contact your security administrator. Then click **Next**.
- 9 Specify the password to use to authorize access to the administrator certificate. This certificate is created automatically by the security upgrade utility. Your security administrator can log in to create certificate files for other users. Click **Next**.
- 10 Specify a **Host** and **Port**. **Port** is already populated with the default RMI Registry port number for the security server.
- 11 Click **Next**.
The upgrade tool publishes the security resources to the security server. If the security server is not running, then an error message opens. After the security server is running, you can try again.
If the site information is already on security server, then a message box opens to inform you that the site already exists on the security server. The security upgrade leaves the existing site. If required, then use the ACL/Role Manager to change permissions or remove the site.
- 12 Restart your computer to complete the upgrade.
To use advanced security, you must obtain a license key.
See [After upgrading security](#).

Upgrading from basic to advanced security

- 1 At the command prompt for your operating system, change to the directory where you installed the system, for example, C:\cloverleaf.

- 2 Specify `hciupgradeutility`, or launch the security upgrade utility from **Start > All Programs > Infor Cloverleaf Integration Suite > Tools > Security Upgrade**.
- 3 Click **Continue with Upgrade**.
- 4 Select **Advanced Security** and click **Next**.
- 5 Select whether to install Certificate Manager. If the only host server is the one on this computer, then you must install the Certificate Manager on this computer. If multiple host servers are on your system, then install the Certificate Manager only on one computer. If you already have host server certificates, then Setup opens a dialog box asking whether to use the existing certificates or to generate new ones. Select **Yes** to install the security management tools (Certificate Manager).
Select **No** to omit the installation of the Certificate Manager. If multiple host servers are on your system, then click **No** for every host server except one.
The upgrade tool then publishes the security resources to the security server.
- 6 Restart your computer to complete the upgrade.
To use advanced security, you must obtain a license key.
See [After upgrading security](#).

After upgrading security

After a security upgrade you must obtain a license key and checksum number.

To do this:

- 1 Log in to <https://support.infor.com> or <https://concierge.infor.com>.
- 2 Select **Resources > Request a Software Key**.
- 3 Complete the required information and submit. A notification is sent stating the key request has been received. Processing is completed within 24 hours.
- 4 After the license is obtained, place the `license.dat` license file into the `vers` directory of your Cloverleaf install directory at `%HClROOT%/vers`.
- 5 Run `hcilicetst` to check if the license is valid.

Note: This applies only to the system license. If you have an Enterprise license, then it is not necessary to regenerate the license after an upgrade. For system license holders, the advanced security add-on module is required after upgrading.

After you receive the key:

- 1 At the command prompt for your operating system, change to the directory where you installed the system, for example, `C:\cloverleaf`.
- 2 Specify `hcilicset`.
- 3 Specify your license key.
- 4 Specify the expiration date of the license. Unless you have received alternative instructions from your Account Manager, press the **Enter** key. This selects the expiration date that is assigned to the license by Infor.
- 5 Specify the checksum number. `hcilicset` verifies the specified information and creates the license key.
- 6 Exit the command window.

Changing the security server

- 1 Click **Change Security Server Only** on the **Security Upgrade** dialog box to change the host and port and publish the security resources into another security server.
The **Security Server Host and RMI Registry Port** dialog box opens with the original host and port.
- 2 Change the host or port as required to publish the security resources to another security server.

Removing the security server

- 1 Save the customer CA certificate files.
- 2 Save the `license.dat` file found in the `\vers` subdirectory of your system root directory. In UNIX/Linux, this is `/vers`. This file contains the license keys required to run security server on this computer, and can be used for future installations. Instead of running `hcllicset`, copy the `license.dat` file to the new `\vers` subdirectory after installation.
- 3 Remove security server.
For Windows, use **Add/Remove Programs** in the Control Panel to remove security server. Optionally, you can delete any extra folders or files.
For UNIX/Linux, run the `uninstall` script.

Downgrading security

When basic security is installed, you can downgrade to no security. When advanced security is installed, you can use the security upgrade utility to downgrade to basic or no security.

You can downgrade security only from a computer that has the host server and client.

Note: When there are multiple host servers, you should perform parallel downgrades at each computer with a basic security.

The procedure for any security downgrade is essentially the same.

- 1 Stop all threads, the Monitor Daemon (`monitord`), and the engine. If necessary, then use the Windows Task Manager.
- 2 Exit and log out all GUIs.
- 3 At the command prompt for your operating system, change to the directory where you installed the system. For example, `C:\cloverleaf`.
- 4 Specify `hciupgradeutility`.
This starts the security upgrade utility and opens the first dialog box. This lists the sites in this system and shows the current security level.
You can also launch the security upgrade utility from **Start > All Programs > Infor Cloverleaf Integration Suite > Tools > Security Upgrade**.
- 5 Click **Continue with Upgrade**.
- 6 Click **no security**.

7 Click **Next.**

The security upgrade utility performs the downgrade and then opens a dialog box notifying you that the security change is complete.

8 Click **OK.****9 Restart your computer or Windows to complete the downgrade.**

Security server administration

There are three levels of Cloverleaf security:

- **No security:** All system network traffic is in clear text. Access to system resources is determined by the network's security.
- **Basic security:** Network communications between clients and host servers are encrypted and transmitted through Secure Sockets Layer (SSL).
- **Advanced security:** Network communications are encrypted and authenticated as they are with basic security. Advanced security builds on basic security to provide fine-grained control over access to system resources.

Security tasks can include:

- Maintaining you security configuration
- Default accounts
- Error messages
- Change security level
- Nodes and ACLs
- Logging
- ACL/Role Manager
- Users, roles, and permissions

Advanced security

Advanced security is an extension of basic security . Basic security enables system security administrators to issue user certificates. These certificates authenticate the recipients as authorized system users. These users can access an organization's system and prevent others from doing so.

If advanced security is installed, then it also has basic security.

If it has only basic security or no security, then the system can be upgraded to advanced security.

Advanced security and basic security both use Secure Sockets Layer (SSL) to implement a secure connection for system configuration. The data connection of a running system engine remains unsecured.

Advanced security, which requires a security server, gives complete and detailed control over access to the message brokering system. With advanced security, you can set user-specific permissions. These define not

only which system resources each user can access, but also the operations each user can perform on each resource.

You can also create roles to which multiple users can belong, and set role-level permissions that apply to all role members. Users can belong to any number of roles. Roles can also belong to other roles, and thus share their permission settings.

Nodes at the lowest level, such as the nodes for individual GUIs, are contained in higher-level nodes, such as the nodes for sites and roots.

The ACL entry for a user or a role lists the permission settings for that user or that role.

An ACL can be created at any level of the node hierarchy, and it automatically applies to all the descendants of that node.

If different permissions are set for a descendant node, then those settings override the inherited settings. This applies to that node and all its descendants.

This combination of user/role options and hierarchical node permissions enables you to create a security structure that is as flexible and complex as you require. You can create an ACL that covers broad parts of the whole structure, yet fine-tune access to individual system resources. You can make any changes that are required.

The effectiveness of the system security depends on the security of the underlying operating system.

Any user with command line access to the host server must also have a unique log-in to the computer system where the host server runs. If users can share operating system level log-ins, then they can impersonate each other and subvert system security.

Users with access to a remote client can use the shell window to gain access to files on the computer system running the host server.

Safeguarding computer systems from these hazards and others is the responsibility of the individual organization.

"administrator" role

In the CLWizard integrated with Infor Ming.le, a Cloverleaf "administrator" role is created after upgrading to advanced mode on security server.

The CLWizard integrated with Infor Ming.le requires the "Infor IFS System" role to map to the Cloverleaf administrator role when the application starts.

Licenses

In addition to the licenses for the engines' host servers, a separate license is required to set up advanced security on each host server. Another license is required for the security server.

Components of advanced security

The Derby embedded database provides a lightweight and internal ACL storage that reduces maintenance overhead.

Advanced security depends on these components:

- One or more system engines
- One host server for every engine
- Multiple clients
- One and only one security server

System engine

The system engine is the heart of the message brokering system. The engine receives messages from various applications, translates those messages from one format to another as required, and transmits them to other applications.

Engine operations can be controlled locally by a command-line interface.

Host server and clients

The host server interacts with the system engine on the same computer system, enabling users to configure, control, and monitor engine processes. Much of the functionality of the host server overlaps command-line functionality.

The host server enables clients on the same computer system, and on remote computer systems, to control engine operation. The host server also enables administrators to audit usage of the GUIs, and provides the foundation for implementing both basic security and advanced security.

Clients are the GUIs that interact with the host server. Clients are installed along with the host server on the computer system that runs the engine. Client can also run on remote computer systems.

Clients provide individual system users with a wide variety of functionality, enabling them to:

- Configure the various message formats supported by the engine.
- Configure the routing and translation of messages within the engine.
- Start and stop engine processes or the individual threads within processes.
- Monitor the activities of the engine.
- Test the processing of messages by the engine.

Security server

The security server is used only in a system that has advanced security. This component runs on a separate computer system from any connected host servers, and interacts with them. As users run clients, the clients request the host servers to perform various actions.

When it receives such a request, a host server queries the security server to determine whether the user has permission to perform the requested action. Depending on the security server's response, the host server performs the action or notifies the user that the request has been denied.

The security server module provides the authentication and authorization services for all the clients, including the host server.

This module gets the ACL information through the Security Store module. It loads the appropriate Security Store when the security server is started.

The security store module functions as the client to the ACL storage.

This module accesses the Derby database using the embedded Derby JDBC driver. The Derby engine runs inside the same Java Virtual Machine as the security server and Derby becomes part of the security server.

The Security Store module contains these parts:

- `SecurityStoreImpl`: This acts as the portal for the Security Store module. This wraps the underlying access implementation to the ACL storage.
- DB Security Store: This runs as the database client and provides all the access methods to the Derby database.

Building a system with advanced security

- 1 Install the host server and the client on one or more computer systems.
- 2 Optionallly, install the client on systems that are connected to the system with the host server.
- 3 On one and only one of the computer systems running the host servers, set up advanced security with Certificate Manager and ACL/Role Manager.
- 4 On the other host server computers in your system, set up advanced security without Certificate Manager and ACL/Role Manager.
- 5 Issue the certificates that users require to access your system.
- 6 Create any roles that apply to multiple users, and build the ACLs that define the permissions that are granted to roles and individual users.
- 7 Distribute the certificate files to your system users, along with the passwords that they require to log in to the GUIs.

After you have built your system, you can perform any necessary modifications, such as revoking user certificates and editing ACLs and roles. You can also monitor security-related activities by reviewing the Audit Log.

Configuring ACLs in Advanced Security mode

ACL (Access Control List) is used to control the permissions of users on advanced mode. The administrator configures this using the ACL/Role Manager.

ACL is a tree structure that consists of these nodes:

- Root
The host.
- Host

The machine name of the host server.

- Version
- Example: integrator20.1.

The ACL can have many hosts and each host can have more than one version. Each node represents a resource for application, command, or configuration files.

Resources and permissions

There are root-level and site-level resources. In most instances, when a user has `config` permission, `execute` permission of an application are also required. `read`, `write`, `insert`, and `delete` permissions of a specific `config` are also required.

Some configurators require additional `execute` permission for a specific `command`, such as `NetConfig` and `Net Monitor`.

Root-level resources consist of:

- `application`. This consists of administrator tools such as:
 - `clapi`
 - `hciaclrolemgr`
 - `hcibox`
 - `hcicertmgr`
 - `hciguiseinit`
 - `hciserveradmin`
- `command`:
`usercmd` is a user command that is not listed in the IDE commands
- `config`:
 - `box` is the Box Manager
 - `eo` is Engine Output Alias
 - `json` is the JSON configurator
 - `proc` is the TCL Proc dialog box
 - `rootInfo` is the `rootInfo`
 - `userfile` is the uploaded file

Site-level resources consist of:

- `application` is for applications such as:
 - `hciaccess`. This opens the IDE
 - `hcialertconfig`. Alert Configurator
 - `hcinetconfig`. Network Configurator
 - `hcinetmonitor`. Network Monitor.
- `command` are commands for actions such as:
 - `hcicmd` for Network Monitor
 - `config` for the configuration files, such as `alert`, `frl`, `hl7`, and `NetConfig`.

Permissions consist of:

- execute. This is for application and command.
- read/write/insert/delete is for config.
 - read opens files as read-only.
 - write saves files.
 - insert creates files.
 - delete deletes files.

ACL configuration

Note: All actions must have "open" action permission.

ACL configuration is available in these areas:

- [IDE](#) on page 1217
- [Administration tools](#) on page 1217
- [Runtime](#) on page 1218
- [Configuration](#) on page 1222
- [ACL example](#) on page 1226

IDE

The IDE action is:

Action	Resource/Permissions	Description
open	site/application/hciaccess - run	Opening a site requires hciaccess permission.

Administration tools

The administration tools and their actions/permissions are:

Tool	Action	Resource/Permissions	Description
ACL/Role Manager	open	application/hciacrolemgr - run	
Audit Log Viewer	open	application/hciauditlog - run	
Certificate Manager	open	application/hcicertmgr - run	
	admin-lite	config/passwords - read	Issues and renews certs without knowing the CA password, if it is already configured on the Server Administration GUI.
Site Init	open	application/hcisteinit - run	

Tool	Action	Resource/Permissions	Description
Server Admin	open	application/hcserveradmin - run	Configures the CA password on the Server Administration for admin-lite.
	Enable CA Password	config/passwords - write	

Runtime

The runtime tools and their actions/permissions are:

Tool	Action	Resource/Permissions	Description
Connection Dump	open	site/application/hciconndisplay - run	
		site/config/NetConfig - read	
	run	site/command/hciconndump - run	
	all	site/application/hciconndisplay - run	
		site/config/NetConfig - read	
		site/command/hciconndump - run	
Database Administrator	open	site/application/hcidbadmin - run	
		site/config/NetConfig - read	
	error database re-send	site/command/hcicmd/resend_errordb - run	
	all	site/application/hcidbadmin - run site/command/hcicmd/resend_errordb - run site/config/NetConfig - read	
Error Database	open, search	site/application/hcierrordb - run	
		site/command/hcicmd/resend_errordb - run	
	resend, edit	site/config/NetConfig - read site - read	

Tool	Action	Resource/Permissions	Description
Network Monitor	all	site/application/hcierrordb - run site/command/hcicmd/resend_errordb - run site/config/NetConfig - read site - read	All other actions require the permission of the "open" action.
	all	site/application/hcinetmonitor - run site - read site/command/hcimsiutil - run site/command/hcisitectl - run site/command/hcicmd - run site/command/hciengineerun - run site/command/hcienginestop - run site/command/hciengineerestart - run site/config/multiview - read and write site/config/NetConfig - read	
	open	site/application/hcinetmonitor - run site/command/hcisitectl - run site/config/multiview - read site/config/NetConfig - read	
	start thread, start with sticky hold, start with hold	site/command/hcicmd/pstart - run	
	stop thread	site/command/hcicmd/pstop - run	
	stop process	site/command/hcienginestop - run	
	restart thread	site/command/hcicmd/prestart - run	
	hold	site/command/hcicmd/phold_obd - run	
	release	site/command/hcicmd/prls_obd - run	
	hold reply	site/command/hcicmd/phold_obd_reply - run	
	release reply	site/command/hcicmd/prls_obd_reply - run	

Tool	Action	Resource/Permissions	Description
	start save	site/command/hcicmd/save_start - run	
	stop save	site/command/hcicmd/save_stop - run	
	cycle save	site/command/hcicmd/save_cycle - run	
	resend	site/command/hcicmd/resend - run	
	reload	site/command/hcicmd/reload - run	
	purge caches	site/command/hcicmd/purgex - run	
	start forward	site/command/hcicmd/fwd_start - run	
	stop forward	site/command/hcicmd/fwd_stop - run	
	start reply forward	site/command/hcicmd/fwdr_start - run	
	stop reply forward	site/command/hcicmd/fwdr_stop - run	
	cycle output	site/command/hcicmd/output_cycle - run	
	engine log configuration	site/command/hcicmd/eo_alias - run	
	reindex	site/command/hcicmd/reindex - run	
	open in-bound SMAT,	site/application/hcismatdb - run	
	open out-bound SMAT	site/command/usercmd - run site/config/smat - read	
	reset statistics	site/command/hcimsiutil - run	
	start process, start with hold, start with sticky hold	site/command/hciengineerun - run	
	restart process	site/command/hciengineerestart - run	
	start monitor, stop monitor	site/command/hcisitectl - run	

Tool	Action	Resource/Permissions	Description
Remote Com- mands	run	site/application/hciremotecmd - run	Remote commands tool requires the permissions of the executing com- mand
Site Daemon	open	site/application/hcsitedaemon - run	"View Logfile" requires the read permission of the site.
	run	site/application/hcsitectl - run site - read	
	all	site/application/hcsitedaemon - run site/application/hcsitectl - run site - read	
SMAT	open	site/application/hciguismat - run site/config/smat - read config/NetConfig - read	The "resend" and "delete" actions require the permission of the "open" action.
	resend	site/command/hcicmd/resend - run	
	delete	site/config/smat - write	
	all	site/application/hciguismat - run site/config/smat - read and write site/config/NetConfig - read site/command/hcicmd/resend - run	
SMAT Database	open	site/application/hcismatdb - run site/command/hcismatcrypt - run site/config/smat - read	
	resend	site/config/NetConfig - read site/command/hcicmd/resend_db - run	
	search	site/command/usercmd - run	
	all	site/application/hcismatdb - run site/command/hcismatcrypt - run site/config/smat - read site/config/NetConfig - read site/command/hcicmd/resend_db - run site/command/usercmd - run	

Configuration

The configuration tools and their actions/permissions are:

Tool	Action	Resource/Permissions	Description
Alert Configurator	open	site/application/hcialertconfig - run site/config/alert - read site/config/NetConfig - read	Requires read permission of NetConfig.
	save	site/config/alert - write	
	new	site/config/alert - insert	
	all	site/application/hcialertconfig - run site/config/alert - read, write and insert site/config/NetConfig - read	
Box Manager	open	application/hcibox - run config/box - read	Opens BOX Manager.
	save	config/box - write	Deploy BOX needs other permissions such as "site"/"config"/"NetConfig".
	import/deploy box	config/box - insert site/config/NetConfig - read	
	delete box	config/box - delete	
	all	application/hcibox - run config/box - read, write, insert and delete	
Database Schema Configurator	open	site/application/hcidbschema - run site/config/databaseschema - read	
	save	site/config/databaseschema - write	
	new	site/config/databaseschema - insert	
	all	site/application/hcidbschema - run site/config/databaseschema - read, write and insert	
Engine Output Configurator	open site eo config	site/application/hcieoconfig - run site/config/eo - read	

Tool	Action	Resource/Permissions	Description
	save site eo config	site/config/eo - write	Site level EO alias requires site configuration permission.
	new site eo config	site/config/eo - insert	Root-level EO alias requires the root configuration permission.
	read root eo config	config/eo/root - read	Master-level EO alias requires root configuration permission.
	save root eo config	config/eo/root - write	
	new root eo config	config/eo/root - insert	
	read master eo config	config/eo/master - read	
	all	site/application/hcieoconfig - run site/config/eo - read, write and insert config/eo/root - read, write and insert config/eo/master - read or site/application/hcieoconfig - run site/config/eo - read, write and insert config/eo - read, write and insert	
FRL Configurator	open	site/application/hcifrlconfig - run site/config/frl - read	
	save	site/config/frl - write	
	new	site/config/frl - insert	
	all	site/application/hcifrlconfig - run site/config/frl - read, write and insert	
Global Variables	open	site/application/hcigvconfig - run site/config/GlobalVariables - read	
	save	site/config/GlobalVariables - write	
	all	site/application/hcigvconfig - run site/config/GlobalVariables - read and write	

Tool	Action	Resource/Permissions	Description
HL7 Configurator	open	site/application/hcihl7config - run site/config/hl7 - read	
	save	site/config/hl7 - write	
	new	site/config/hl7 - insert	
	all	site/application/hcihl7config - run site/config/hl7 - read, write and insert	
HPRIM Configurator	open	site/application/hcihprimconfig - run site/config/hprim - read	
	save	site/config/hprim - write	
	new	site/config/hprim - insert	
	all	site/application/hcihprimconfig - run site/config/hprim - read, write and insert	
HRL Configurator	open	site/application/hcihrlconfig - run site/config/hrl - read	
	save	site/config/hrl - write	
	new	site/config/hrl - insert	
	all	site/application/hcihrlconfig - run site/config/hrl - read, write and insert	
Import Script		site/application/hciimport	
Json Configurator	open	site/application/hcijsonconfig - run site - read	
	save	site - write	
	new	site - insert	
	all	site/application/hcijsonconfig - run site - read, write and insert	JSONACL has issues. This should be "site"/"config"/"json".
LDL Configurator	open	site/application/hcildlconfig - run site/config/ldl - read	
	save	site/config/ldl - write	
	new	site/config/ldl - insert	

Tool	Action	Resource/Permissions	Description
Lookup Table Configurator	all	site/application/hcildlconfig - run site/config/ldl - read, write and insert	
	open	site/application/hcitblconfig - run site/config/table - read	
	save	site/config/table - write	
	new	site/config/table - insert	
	all	site/application/hcitblconfig - run site/config/table - read, write and insert	
Network Configurator	open	site/application/hcinetconfig - run site/config/Multiview - read site/config/NetConfig - read site/config/siteInfo - read	
	save	site/config/Multiview - write site/config/NetConfig - write	
	all	site/application/hcinetconfig - run site/config/Multiview - read, write site/config/NetConfig - read, write site/config/siteInfo - read	
Script Editor	open	site/application/hciguieditor - run site/config/proc - read	
	save	site/config/proc - write	
	new	site/config/proc - insert	
	all	site/application/hciguieditor - run site/config/proc - read, write and insert	
Translation Configurator	open	site/application/hcitranslateconfig - run site/config/xlate - read	
	save	site/config/xlate - write	
	new	site/config/xlate - insert	
	all	site/application/hcitranslateconfig - run site/config/xlate - read, write and insert	
UN/EDIFACT Configurator	open	site/application/hciedifactconfig - run site/config/edifact - read	

Tool	Action	Resource/Permissions	Description
VRL Configurator	save	site/config/edifact - write	
	new	site/config/edifact - insert	
	all	site/application/hciedifactconfig - run site/config/edifact - read, write and insert	
	open	site/application/hcivrlconfig - run site/config/vrl - read	
	save	site/config/vrl - write	
	new	site/config/vrl - insert	
	all	site/application/hcivrlconfig - run site/config/vrl - read, write and insert	
	open	site/application/hcix12config - run site/config/x12 - read	
X12 Configurator	save	site/config/x12 - write	
	new	site/config/x12 - insert	
	all	site/application/hcix12config - run site/config/x12 - read, write and insert	
	open	site/application/hcixmlpm - run site/config/xml - write	
XML Package Manager	new	site/config/xml - insert	
	delete	site/config/xml - delete	
	all	site/application/hcixmlpm - run site/config/xml - write, insert and delete	
	open/save	site/application/hcixmlpm - run site/config/xml - write	

ACL example

To open the Alert Configurator of a site named "helloworld":

- Replace "site" in the resource path with the site name.
- Grant run permission for helloworld/application/hcialertconfig.
- Grant read permission for helloworld/config/alert.
- Grant read permission for helloworld/config/NetConfig.

Security modes

This table describes the different security modes:

Security level	Description
No security	<p>All system network traffic is in clear text. Access to system resources is determined by the network's security.</p> <p>A system with no security must have at least one engine and a host server for every engine, and may have any number of clients.</p>
Basic security	<p>Network communications between clients and host servers are encrypted and transmitted through SSL (Secure Sockets Layer). The organization's Security administrator issues a unique revocable certificate to every authorized user. With a valid certificate, a user can access all host server functionality. Without it, the user has no access to the system at all.</p> <p>Engine communications over the network are transmitted in clear text.</p> <p>A system with basic security must have at least one engine and a host server for every engine, and may have any number of clients.</p> <p>The Certificate Manager runs in local mode, as opposed to running as a remote client, on one and only one host server. This provides centralized certificate administration for all host servers in the system. All User certificates are issued with Certificate Manager, and then manually copied to the computer systems that run the client.</p>

Security level	Description
Advanced security	<p>Network communications are encrypted and authenticated as they are with basic security. Advanced security builds on basic security to provide fine-grained control over access to system resources. Used in conjunction with Certificate Manager, the ACL/Role Manager enables a security administrator to assign different permissions to different users.</p> <p>A security administrator can create roles for multiple users. By assigning permissions to those roles, administrators can assign the same permissions to all users at once. Role members can be other roles and users.</p> <p>Typically, a system with advanced security has multiple engines and the host servers to control them, along with multiple clients to provide remote user interfaces. One and only one of the host servers should have both Certificate Manager and ACL/Role Manager. All of the host servers in the whole system are connected to one and only one security server on a separate computer system.</p> <p>This configuration gives the security administrator complete centralized control over the whole system. If certificates were managed on multiple host servers, then the security administrator's task would be much more difficult.</p>

Certificates

Certification is the first step in securing a system. Each user must have a certificate to authenticate access to the system. Each certificate consists of two related files: a public certificate file and a private key file. The private key file can only be accessed by specifying a password, which must be issued to the user who is identified by the certificate.

Certification begins with a CA or Certificate Authority. This is the entity that issues certificates and vouches for the information that they contain. Infor is the ultimate CA for a system. All system certificates must be generated from within the system and reference Infor as the ultimate CA. Certificates are not recognized from other CAs such as Verisign.

Infor issues customer CA certificates to organizations that purchase the system with basic and advanced security.

The customer CA certificate makes the purchasing organization a CA and empowers it to issue its own user certificates and server certificates.

User certificates, also known as client certificates, are copied to computer systems that run clients. The associated passwords are given to the users of those computer systems.

Server certificates are set up during installation of host servers and the security server, and invoked automatically without user intervention. Then they are invoked whenever the requirement arises.

Infor gives the organization a unique CA public certificate file, a unique CA private key file, and a CA password. These are specified during security setup. That same CA password is given to the security administrator to authenticate access to Certificate Manager.

When a user logs on to a GUI, certificates are used to establish two SSL connections. The first one is between a client and a host server, and the second is between that host server and the security server.

The system verifies that the user certificate is one that was issued by the customer CA. It also verifies that the customer CA certificate is one that was issued by the Infor CA.

After the login is complete, the user is granted the permissions that are associated with the user certificate.

The security administrator must safeguard the private key file. Public certificate files and public key files may be exchanged freely, but exchanging a private key file with anyone can compromise an SSL-secured connection. Therefore, Infor never distributes its CA private key file.

Starting with advanced security

The general sequence for starting the system with security server is:

- 1 Start the security server.
- 2 Start the host server.

After the servers have been started, the system with advanced security is ready for access by the client GUIs.

Running with advanced security

Secure communications must be maintained in a system with advanced security. Client GUIs should be accessed only by logging on to the system from computer systems where clients have been installed.

In this way, all client/server communications are transmitted through SSL.

Stopping with advanced security

Before you stop the system with advanced security, ensure that all system users have logged off their client GUIs. If the system is stopped when any GUIs are running, then all modifications in progress are lost.

To stop a system with advanced security, you must stop the security server and the host server to which it is connected. You can perform these tasks in any order.

Stopping the security server

In UNIX, use of these methods to stop the security server:

- Run the `kill` command on the `hcisecurityserver` process ID.
- From the command line, run `hciss -k s`.

In Windows, use of these methods to stop the security server:

- Use the Task Manager to stop the `hcisecurityserver` process.
- From the command line, run `hciss -k s`.

Note: You can stop the security server without stopping the host server.

Stopping the host server

In UNIX, use either of these methods to stop the host server:

- Run the `kill` command on the `hcihostserver` process ID.
- From the command line, run `hciss -k h`.

In Windows, use either of these methods to stop the host server:

- Use Task Manager to stop the `hcihostserver` process.
- From the command line, run `hciss -k h`.

Verifying a server is running

In UNIX, you can verify that the host server or the security server is running with the `process status` command (`ps`) and `grep` for `hci`.

In Windows, verify that the host server or the security server are running by checking Task Manager for these:

- System service: `hciservice`
- Security server: `hcisecurityserver`
- Host server: `hcihostserver`

In UNIX or Windows, you can verify that the host server or security server is running with of these methods:

- From the command line, specify `hciss .`
- Change to the `\clgui\bin` subdirectory of your system root directory and run this command `hcireglist host port`.
 - `host` is the name or IP (Internet Protocol) address of the computer system. To identify the local computer, use `hostname`.
 - `port` is the port number of the port that the host server uses for TCP/IP communications. By default, the port number is 13021.

For example, `hcireglisthostname13021` enables the `hcireglist` utility. This is included in the host server installation.

This utility lists the contents of an RMI registry for a specific port on the computer system. RMI is the communications protocol that provides communication between the security server, host server, and client. If `hcireglist` lists the RMI registry for a security server or a host server, then that server is running.

Although the `hcireglist` utility is included only in host server installations, it can be copied to the security server or to clients. It must be placed into the `\clgui\bin` subdirectory of your system root directory.

Output examples

These examples show different sets of partial output that indicates whether the host server or security server is currently running.

- Example 1

This example shows what you see when neither server is running.

```
...
Unable to list registry
Connection refused to host: [hostname:13010]; nested exception is: java.net.ConnectException:
Connection refused
...
```

- Example 2

This example shows what you see when the security server is running. In this example, the first line is a listing of the contents of the registry.

```
SecurityServer_1.0
SecurityServer_1.0
com.hie.securityserver.SecurityServerImpl_Stub
[RemoteStub [ref: [endpoint:
[208.20.150.35:1585](remote),objID:[2c046e:e0562b08b9:-8000, 1]]]]
...
```

The remaining entries are a lookup of that registry object.

- Example 3

This example shows what you see when the host server is running.

```
...
RMI_CloverleafServer_1.0
RMI_CloverleafAuditServer_1.0
RMI_CloverleafServer_1.0
com.hie.Cloverleaf.remoteserver.RemoteCloverleafServer_Stub
[RemoteStub [ref: [endpoint:
[208.20.150.35:1606](remote),objID:[2c0e21:e0562da1dc:-8000, 1]]]]
RMI_CloverleafAuditServer_1.0
com.hie.securityserver.AuditServerImpl_Stub
[RemoteStub [ref: [endpoint:
[208.20.150.35:1606](remote),objID:[2c0e21:e0562da1dc:-8000, 1]]]]
...
```

In this example, the first two lines list the contents of the registry. The remaining entries are lookups of the two registry objects.

Advanced security logging

A system with advanced security reports various events in these log files:

- `\cis6.2\integrator\security\logs\security.log`
This is the error log generated by the security server. Restarting the security server creates a new `security.log` file and places the contents of the old log into `security.log_bak`.
- `\cis6.2\integrator\server\logs\server.log`
This is the error log generated by the host server. Restarting the host server creates a new `server.log` file and places the contents of the old log into `server.log_bak`.
- `\cis6.2\integrator\server\audit\audit.log`
This is the error log generated by the host server that contains information used by Audit Log Viewer (`hciauditlog`). This log continues to grow until deleted.
- Immediately after a successful startup of the security server, the log is located at `\integrator\security\logs\security.log`.
- Immediately after a successful startup of the host server, the log is located at `\integrator\server\logs\server.log`.

Error messages in the log files

Error messages are listed in the log files. Troubleshooting information is provided for errors such as:

- License errors
- RMI errors
- Audit failures
- Client GUIs
- Log-in mismatch
- Insufficient security permissions

License errors

An error message prompts if you do not have the appropriate license for a feature.

For example:

```
cssrvr:main: ERR: SecurityServer initialization failed:  
Invalid CLOVERLEAF Integration Services ADVANCED security license
```

Contact your Account Manager to obtain the correct license.

RMI (Remote Method Invocation) errors

Any error message about RMI is a message stating that RMI is already bound, not found, and so on. This indicates that the system components, for example, security server, host server, or client, cannot communicate with each other.

This may happen because:

- One of the system components is down.
- Two instances of the same system component have been invoked.
- Another application, such as Microsoft Outlook, has taken control of the RMI port.

The system components communicate through RMI.

The workaround is to ensure that all system components are shut down. Then, you can verify that default RMI port 13021, is not in use. To do this, specify `netstat -a` from the command line. After you ensure that port 13021 is available, restart the components.

Audit failure errors

Error messages relating to audit failure happen when the security mode is changed or when the system software is upgraded to a new version. For example, when basic security is upgraded to advanced security.

The best solution is to delete the `\cisversion\integrator\server\audit\audit.log` file, and then restart the host server.

Client GUI errors

If an error message indicates that a client GUI cannot launch, then the host server might not be running.

Another possibility is that the user ID specified at login did not match the user certificate.

Note: User IDs are case-sensitive.

Log-in mismatch errors

You can resolve a log-in mismatch by deleting the caches where log ons and the environments that they access are saved.

- 1 Delete all the `.lsc` files from `\cisversion\integrator\client`. (These are the credential cache files used during log-in.)
- 2 Save `\cisversion\integrator\client\client.ini`, and create a new one.
- 3 Add these lines to the new credential cache:

```
[general]
doc_base_dir=E:\root\integrator\
debug=false
```

The `doc_base_dir` line is for online documentation. Here, *root* is the root directory. For example, *cisversion*.

- 4 Save the file, and restart the client GUI.

Insufficient security permissions errors

Any error message indicating that security permissions are insufficient can happen for a wide variety of reasons.

The first step in resolving such an error is to run the ACL/Role Manager and verify that the user has been assigned the required permissions. If this is not the problem, then use this procedure:

- 1 Change these settings in `security.ini`:

Change from:

```
[logging]
CLOVERLEAF_security_server_category=info
CLOVERLEAF_security_server_level=brief
security_server_category=info
security_server_level=brief
```

Change to:

```
[logging]
CLOVERLEAF_security_server_category=debug
CLOVERLEAF_security_server_level=verbose
security_server_category=debug
security_server_level=verbose
```

- 2 Restart the security server.
It is not necessary to restart the host server.
- 3 Launch the denied GUI.
- 4 Capture the error log file generated by the security server and send it to Support. This is located at `\cisversion\integrator\security\logs\security.log`.
- 5 Reset the `security.ini` settings.
Note: Debug mode captures a large amount of data.
- 6 Restart the security server again.
It is not necessary to restart the host server.

Changing modes of security

- 1 Change the security mode for testing by editing the `[security]` section of the `\cisversion\integrator\server\server.ini` host server configuration file.

The correct `[security]` sections for the various modes are:

- No security:

```
[security]
security_server_used=false
basic_security_enabled=false
```

- Basic security:

```
[security]
security_server_used=false
basic_security_enabled=true
```

- Advanced security:

```
[security]
security_server_used=true
basic_security_enabled=true
```

- 2 After modifying the `server.ini` file, restart the host server.
- 3 Delete the `\cisversion\integrator\server\audit\audit.log` file.
In advanced security, this file has a digital signature. If the digital signature is not correct, then it can prevent the host server from starting successfully.

Removing a site in advanced security

- 1 Verify that all users are logged off the system.
- 2 On the **ACLs** tab of the ACL/Role Manager,, right-click the site node and then select **Remove** in the resulting context menu.
- 3 On the machine that runs host server, modify the **Environ** setting in the `\cisversion\integrator\server\server.ini` file to remove the listing for the site.
- 4 Delete the site directory and all its subdirectories.
- 5 Restart the host server.

Nodes and ACLs

A system is broken down into nodes, each of which is a system function or set of functions. These nodes are arranged in a branched hierarchical structure, similar to a directory tree.

The top or trunk node, called host, is a container for all the functions in the system. The branches of this node are containers for the functions of individual host servers. Within those containers are nodes that are containers for individual roots. There is one in a Windows system and one or more in a UNIX system.

Within the root nodes are nodes for functions that are available at the root level. There are also nodes that are containers for the functions of individual sites within the roots.

In a system with advanced security, each node has its own ACL (Access Control List).

An ACL lists the users who are granted access to the node. It also defines specific permissions for each user. A permission determines which operations that user can perform at that node.

Similar to nodes, ACLs are hierarchical. Thus, permissions set at any high-level node are automatically extended to all the nodes contained within that node. For example, when advanced security is set up, an ACL is defined for the application node within the system root node. Then, the system security administrator can perform security-related functions.

The permissions set in the ACL for the application node are automatically extended to all the nodes within the application. This also applies to nodes have no ACLs of their own. For example, `hciguiseinit`, `hciaclrolemgr`, `hcicertmgr`, and `hciauditlog`.

Nodes within nodes can have ACLs of their own. These can be different from the parent ACLs, with different users and different permissions for the same user.

For example, a system administrator is listed in the ACL for the host node and is granted all permissions. That same administrator could be denied all permissions in the ACL for the application node, so that security administration can be independent of server administration.

The parallel hierarchical structure of nodes and ACLs has these advantages:

- It enables the security administrator to control access to each component of the system individually.
- It provides a parent-child framework that simplifies administration. By default, the ACL for any parent node is automatically applied to any child nodes.

With advanced security, system access can be fine-tuned to the exact degree that is required. You can avoid the task of defining ACLs node by node, and define one ACL that applies to a whole family of nodes. Then you can define exceptions for individual nodes within the family.

All database files are stored under `$HCIRoot\security\data\cl_acls`.

Summary of rules for ACLs, roles, and users

These are the major rules that govern the construction of ACLs and the interactions among nodes, roles, and users:

- The ACL for a system node is automatically applied to any other nodes contained within that node. In effect, creating an ACL for a container node automatically creates default ACLs for all of its content nodes, their content nodes, and so on.
- Any ACL explicitly defined for any content node always takes precedence over the default ACL created for higher-level container nodes. For example, the ACL for a parent node grants a permission to a user or role. The ACL, though, for a child node denies that permission. In this instance, the permission is denied.
- Permissions that are not explicitly defined for a node are automatically defined by the closest ancestor node. For example, the ACL for a node does not explicitly grant or deny a permission. That permission is granted by the ACL for the parent node but denied by the ACL for the grandparent node. In this instance, the permission is granted.
- Permission settings for any role are automatically applied to any roles and users that belong to that role. In effect, setting the permissions for a role creates a default template for all the members of that role. For example if Role A belongs to Role B, by default Role A has the same permission settings as Role B.

- In example, one role is a member of another role. Permission settings for a member role take precedence over permission settings for a role to which the member role belongs.
If there are multiple role membership levels, then the closest role in the membership hierarchy takes precedence. For example, Role A belongs to Role B, which in turn belongs to Role C.
In another example, if a permission is not specified for Role A but is granted for Role B, then that permission is granted to users who belong to Role A. This is even if it is explicitly denied for Role C.
Conversely, a permission that is granted for Role C but denied for Role B is also denied for Role A.
- Users who belong to multiple roles are automatically granted any permissions that are granted to any of those roles. This applies even if one or more other roles deny those same permissions.
- Permission settings for a user always take precedence over the permissions settings for the roles to which that user belongs.

Roles as members of other roles

When a role is a member of another role, all permissions that are granted to the second role are extended to all members of the first role. For example, the Quality Tester role is made a member of the System Operator role. You can grant all System Operator permissions to all members of the Quality Tester role.

The Quality Tester role can also be granted permissions in addition to those granted to the System Operator role. This does not affect any other members of the System Operator role. The Quality Tester role inherits permissions from the Operator role. The Operator role does not inherit permissions from the Quality Tester role.

A user who belongs to the Quality Tester role inherits all that role's permissions. They can be modified at the user level without affecting other members of the Quality Tester role.

Multiple roles

Consider the case of a user who requires all the permissions of both a Protocol Engineer and a Translation Engineer. This does not pose a problem. Any user can be a member of as many roles as necessary and thereby acquire all the permissions of all those roles.

Make the user a member of both the Protocol Engineer and Translation Engineer roles. That user is automatically granted all the permissions of both roles. Even if the Protocol Engineer role expressly denies some permissions that are granted to the Translation Engineer role, the user has those permissions. That same user has any permissions that are granted to the Protocol Engineer role, even if those permissions are denied to the Translation Engineer role.

When a user is made a member of a particular role, the person is granted all the permissions that are associated with that role. This is regardless of any permissions that are denied in any other roles to which the user belongs.

Interaction of user- and role-level permissions

It may be necessary to individualize permissions for a given user, even though that user has a basic function similar to that of other users. Advanced security is designed so that role-level permissions and user-level permissions can be combined to define the appropriate level of access for each individual user.

For example, one Translation Engineer works on threads that translate X12 messages to HL7 messages. Another Translation Engineer works on threads that translate FRL messages to HL7 messages. You can set permissions at the `hl7` and `hclhl7test` nodes for the Translation Engineer role.

Then, set user-level permissions for one Translation Engineer at the `x12` and `hcix12test` nodes. After this, you can set user-level permissions for the other Translation Engineer at the `frl` and `hcifrltest` nodes.

Users and roles

Use the ACL/Role Manager to assign user-level and role-level permissions. A role is an arbitrary association based on criteria you select. For example, common tasks that are performed by multiple users, the requirement of certain users for certain types of information, and so on.

Role-level permissions afford you the convenience of setting permissions for whole groups of users at once. By combining user-level permissions and role-level permissions, you can implement a complete security structure with as much compartmentalization as you require.

Implement as many roles as you require on your system, and make as many users as necessary into members of each role.

Roles that might be implemented on a typical system:

- System Administrator
 - Task: Oversees the system configuration and implementation.
 - Permission: All permissions for all nodes except `hclaclrolemgr` and `hclcertmgr`.
- Security Administrator
 - Task: Manages user certification and ACL/role structure.
 - Permission: All permissions for all nodes within the application under the system root node.
- Protocol Engineer
 - Task: Configures protocol threads.
 - Permission: Selected permissions for selected nodes under each site node.
- Translation Engineer
 - Task: Configures translation threads.
 - Permission: Selected permissions for selected nodes under each site node.
- Quality Tester
 - Task: Tests and analyzes system operation.
 - Permission: Permissions to run for all nodes within command under each site node.
- System Operator
 - Task: Starts, stops, and monitors system performance.
 - Permission: Permissions to run for selected nodes within command and application under each site node.

Assign permissions to roles in the system the same way that you assign permissions to individual users. For example, there may be several Quality Testers who must perform the same tasks to test and analyze system performance. Instead of setting the same permissions for each Quality Tester individually, you can set those permissions once for the Quality Tester role. Then, you can make all those users members of that role.

Roles can also be members of other roles. For example, the Quality Tester role might be a member of the System Operator role. All permissions that are granted to the user members of the System Operator role are extended to all user members of the Quality Tester role.

Security control

An ACL permission for security control is located in the CLAPI generic file. Available nodes in ACL/Role Manager are:

- `clapi` under `root/application`.
- `command` and its sub-node `command/usercmd` under `root`.
- `userfile` under `root/config`.

Adding nodes

When you add a host, a root, or a site to the ACL tree, you add only the pertinent parts of the node scheme. For example, adding a root does not automatically add a site.

The parts that are added come from the original template. You do not add any other content nodes that have been added to the original, or copy any ACLs. The new node and all its contents are a blank slate.

When you add a host, you must also add a root, because every host machine must run the host server. This root can have the same node name as a root in another host machine.

Only sites that are created using the Site Init GUI tool are automatically added to the ACL. If you use the `hcisiteinit` command to create the site, then it is not automatically added to the ACL even in advanced security mode.

The only times a site is automatically added to the ACL are:

- Using the Site Init tool to add a site when in advanced security mode.
- Using the `hcisiteinit` command or the Site Init GUI tool in no security or basic security and then upgrading to advanced security.

Nodes for variants of the configuration files for message formats can be added to the ACL tree. This is in addition to hosts, roots, or sites. For example, HRL or FRL.

A few of the nodes for standard message formats, such as HL7 and X12, already contain nodes for the standard variants.

Building ACLs

Use this general process to build ACLs:

- 1 Plan your ACL/role/user structure.
 - Identify all the users who require to be authorized to access your system.
 - Define any roles those users must belong to, and any roles that can have other roles as their members.
 - Determine which permissions should be granted to which users on which nodes.
- 2 Issue User certificates for all the users who are required to authorized to access your system.

Issue certificates to users before adding them to roles or ACLs. After you have added a user with the ACL/Role Manager, you cannot issue a certificate to that user.

Issuing a certificate to a role has no effect.

Do not issue certificates to roles. This subverts system security by having a certificate that is shared by many users, and is inherently insecure.
- 3 Add all the roles that you have defined.
- 4 Modify roles as necessary to specify which roles belong to other roles.
- 5 Modify users as necessary to specify to which roles and ACLs they belong.

Use cases

Examples of use cases include:

- Importing LDAP groups:

As a user, you require importing groups configured on an enterprise Active Directory server to Cloverleaf. From the user interface, select the groups on the Active Directory server that you must import. The selected groups are created in security server. These groups contain appropriate users that are assigned to the groups, to mirror the Active Directory configuration.

Users that are configured on Active Directory, but not in security server, are optionally created in Cloverleaf. You decide whether to create any additional users.
- Synchronizing LDAP groups:

You have made changes to the group configuration on the Active Directory server. Select the **Synchronize Groups** option, so that changes to selected groups are made on the security server.

Any new users are optionally imported as well. Select the option to import new users that are assigned to selected groups.
- Mapping ACL to user:

As a developer, you have received specifications for a new interface. You only have permissions in Active Directory to access the test system. You open the Cloverleaf IDE and log in.

The only remote host visible to you is the test system. Select the host and continue your work.

ACL/Role Manager

The ACL/Role Manager enables the security administrator of a system to implement the granular control offered by security server. With ACL/Role Manager, the security administrator can determine which users have access to which system features, and which operations those users can perform.

You must be at the machine that runs the host server to start the ACL/Role Manager.

In addition, this information is required:

- The user name and password that are assigned to the certificate administrator.
Initially, the name is set to "administrator" by Infor. The password is set during the system installation process.
- The CA password that is assigned to your organization by Infor.

Access is through the IDE or a command prompt:

- From the IDE, open the Shell Window. Then, at the command prompt, specify `hciaclrolemgr` to load the GUI.
- In Windows, select **All Programs > Infor Cloverleaf Integration Suite > AclRoleMgr**.
- At a command prompt (DOS or UNIX), change directories to the `/clgui/bin` directory for your system root. Then, specify `hciaclrolemgr`.

Logging on

After starting the ACL/Role Manager, you must log in.

If you are not already logged on to the system as administrator, or if your login credentials are not cached, then the **Please Login** dialog box opens.

On the **Please Login** dialog box, the **User Name**, **Certificate Path**, and **Private key Path** fields automatically keep the settings from the most recent login operation. You can change these settings as required.

New users must create credentials.

- 1 Ensure that the **User Name** field specifies **administrator**. This is the user name that is defined for the security administrator during system installation.
- 2 Ensure that the **Certificate Path** field identifies the appropriate certificate file. This file is always in the `%user folder%\clide\certs` subdirectory of the directory where the system is installed, and it always has the file name `administrator-cert.der`.
- 3 Ensure that **Private key Path** identifies the appropriate certificate key file. This file is also in the `%user folder%\clide\certs` subdirectory of the directory where the system is installed, and it has the file name `enc-administrator-key.der`.
- 4 Specify the password specified for the administrator user ID in the **Password** field.
- 5 If you plan to use another system GUI, or if you plan to use the ACL/Role Manager more than once, then select **Use Credential Cache**. This automatically applies the log-in information throughout the current system session.
If this check box is cleared, then you must log in manually each time you start a system GUI.
- 6 Click **Apply**. This loads the ACL/Role Manager.

Exiting the ACL/Role Manager

Exit the ACL/Role Manager in one of these ways:

- Click **File > Exit** to start the ACL/Role Manager or any other system GUI without logging back in. You must still supply the CA password to access the ACL/Role Manager again.
- Click **File > Log Off & Exit** to ensure that no system GUI can be started by anyone, including yourself, without logging on.

Exiting has no effect on any other open GUI, even if you do log out. You can continue to use any GUI that is already open.

ACL/Role Manager GUI

The ACL/Role Manager is the central control console for managing users, roles and ACLs.

It is assumed that some users have already been issued certificates to access your system. Remember, users must be issued certificates before they can be members of roles or be assigned permissions in ACLs.

File > Refresh is useful only if another user might make changes to security-related files when you are using the ACL/Role Manager.

Click **Synchronize** to send a synchronization request to the security server. The security server starts synchronization according to the defined synchronization mode. The **LDAP Synchronize** dialog box shows the synchronization progress for the imported users, roles, and hosts.

Users tab

This tab has a tabbed panel that contains:

- **General** tab: This contains a list of all the users who have been issued certificates that enable them to access the current system root, along with information for each user.
- **User ACLs** tab: This is a tree table for the resource tree, which lists all resources and is for configuring access permissions for the current user.

Buttons along the bottom enable you to manage user data:

- New users must be unique. You can add user information in **Remark**.
A user certificate cannot be issued after the user has been added with ACL/Role Manager. Usually, therefore, you should not use **Add New User**. Instead, use Certificate Manager to issue User certificates, which automatically adds the users. **Add New User** is included for use in exceptional circumstances. For example, when a system with basic security is upgraded to advanced security or when you must reinstate a user whose certificate has been revoked.
- **Modify User** is the only way to edit user information; you cannot edit the text boxes on the Users tab. Changing a user name deletes the existing user and then adds the new one.
- After a user has been deleted, that user is not authorized access to the system at all. The user is also removed from any roles.

When an LDAP user is imported, the user's information is displayed on the tab. This includes the user name, DN, Remark, and role.

Tree table icons

The Lock icon indicates a permission cannot be used for the resource. For example, an application or command has no read action permission. This permission is not changeable.

The gray checkmark indicates that the user is granted the permission for the resource through groups or configuration of the parent folder of the resource.

For example, a user belongs to a group, and the group is granted the permission, so the user is granted the permission through the group.

In another example, a parent folder of the resource is configured with the permission for the user. The user is granted the permission for the resource through the parent folder.

- Double-clicking this icon or cell changes this icon to a dark gray “X”. For the user, this adds a denied action permission for the resource.
- Right-clicking this icon opens a menu for permission configuration, with options for **Granted**, **Denied**, and **Not Specified**.

An “X” indicates the user is denied the permission for the resource through groups or configuration of the parent folder of the resource.

- Double-clicking this icon or cell changes this icon to a dark gray checkmark, which for the user adds a granted permission for the resource.
- Right-clicking this icon opens the menu for permission configuration.

A dark gray checkmark indicates that the user is configured with granted permission for the resource directly.

- Double-clicking this icon or cell should change this icon to dark gray “X”. For the user, this adds a denied action permission for the resource.
- Right-clicking this icon opens the menu for permission configuration.

A dark gray “X” indicates that the user is configured with denied permission for the resource directly.

- Double-clicking this icon or cell changes this icon to a dark gray checkmark, which for the user adds a denied action permission for the resource.
- Right-clicking this icon opens the menu for permission configuration.

Roles tab

This tab has a tabbed panel that contains:

- **General** tab: This contains a list of all added roles, with detail information.
- **Role ACLs** tab: This is a tree table for the resource tree, which lists all resources and is for configuring access permissions for the current role.

The tree table has RESOURCE, INSERT, DELETE, READ, WRITE, and EXECUTE columns. The first column displays the resource tree.

The buttons along the bottom enable you to manage role data:

- New roles must be unique. You can add role information in **Remark**.
- **Modify Role** is the only way to edit role information; you cannot edit the text boxes on the **Roles** tab.

After a role has been deleted, it is removed from role and user membership lists.

Resources

A resource can be an application, command, configuration file, application group, file group, and so on. Resource characteristics are:

- For every resource, there are available action permissions: read, write, insert, delete, and run.
- Action permissions are shown with one of five icons.
- Selected cells are moved using the up/down/left/right keys.
- You can also click in a blank space to change the cell value.
- Click **Delete** to remove the configuration for the cell for the current user.
- Pause over a cell to see a tooltip of permission details for the resource.

Show All Resources shows/hides resource entries which are not related to the user.

If the check box is cleared, then the host is not displayed in the resource tree. This is when the user does not have any granted or denied permissions for resources within a host,

If a user does not have any granted or denied permissions for resources in a site, then the site does is not displayed in the tree.

Events tab

This tab contains a paginated table that displays user access events and search criteria controls. This includes fields for user name, host, site, date range, event type, and others.

The “...” button next to a field opens a dialog box for selecting items from a list. You can also specify text in the field.

date range contains two date fields: **start** and **end**.

Clicking the date field opens a chooser dialog box. The default date range is the most recent week from the current day.

ACLs tab

The left pane of the ACLs tab is a tree of the nodes in your system enterprise.

There are two types of boxes that display to the left of a node name:

- A white box icon indicates there is no ACL for the node. The users or roles that have permissions to perform operations are those that are included in the ACL for any higher-level nodes containing this node.
- A yellow box icon indicates there is an ACL for the node, and therefore at least one role or user with permissions.

When a system with advanced security is installed, a user named "administrator" is automatically issued a certificate. This user is added to the ACL for the root's application node. This contains nodes for the various security-related functions.

The right pane shows the ACL information for the selected node. This includes these items:

- Node name.
- Names of all the roles and users authorized to access the node.
- Permissions for each role or user.
- The **Name/Permission** table in the right pane indicates that the user or role has permissions shown by the user or role icon.
- If the user and role have the same name, then the ACLs are shown in different rows. Different icons distinguish between the user/role.

Except for run, the other rights (read, write, insert, and delete) are disabled on the **Permission Modification** dialog box. This happens when the permission is modified on the nodes that delegate the commands or applications. For example, the `command` node under a site node, the `hcicmd` node under the `command` node, the `application` node, or the `hciaccess` node.

The buttons along the bottom enable you to manage ACL data:

- **Add** is where you identify the role or user to be added and select the permissions for that role or user.
- **Modify** is where you can change the permissions for the selected role or user. This is the only way to modify permissions; you cannot edit the text boxes within the Roles or Users tab.

When you delete a role, you automatically eliminate the permissions for all of that role's member roles and users. This does not affect any permissions set for them in other roles, or any associated user-level permissions.

clsecurityaudit node

The `clsecurityaudit` node on the ACLs tab is for security audit permission control.

In None and Basic security mode, there is no permission control for Security Audit.

In Advanced security mode, the `clsecurityaudit` node on the ACLs tab is for generation, export, and view security audit control in the Server Administration dialog box.

Additional support includes:

- read permission of `clsecurityaudit` is for view control.
- run permission of `clsecurityaudit` is for generation and export control.

Context menu for the ACLs tab

Open a context menu for managing the ACL node list by selecting a node and then right-clicking anywhere in the left pane.

Depending on the level of the selected node, the context menu contains these commands:

- **Add Host** is where you specify the name of the remote machine node to be added. Adding a host node automatically adds all the standard component nodes for that host. This is available only at the host node at the top of the system.
- **Add Root** is where you specify the name of the root node to be added to the selected node. This is available only at the node for a host server.
- **Remove Host** is available only at a host server node.
- **Add Site** and **Remove Root** are available only at a system root node.

- **Remove Site** is available only at a site node.
- **Copy Site/Root Permissions** copies the permissions for the selected site/root to another site/root that you specify, overwriting any permissions that already exist.
- **Add Node** and **Delete Node** are available only at nodes for configuring elements within a site.
The only nodes that can be deleted are those that were added with **Add Node**.
- **Permissions Check** lists all the users with permissions for the selected node, along with the permissions for each user.

Admin-lite

The admin-lite role is for those users who do not know the CA password but must create a certificate.

Overview of features:

- A `config/passwords` node is available for password access control on the **ACL/Role Manager**.
- When advanced security is enabled, **Enable CA password** is on the **Server Administration**'s **Run** menu, if you have write permission for passwords. You enable the CA password by specifying it on the **Server Administration** tool.

Disable CA password is on the **Run** menu when the CA password has already been stored.

The CA password is automatically loaded on the Certificate Manager tool if you have read permission for passwords.

Adding the passwords node to the ACL node

On the **ACLs** tab of the **ACL/Role Manager** dialog box, the `passwords` node is added as a child of **host > sitename > integrator19.1 > config**.

Server Administration dialog box

The **Run** menu options for admin-lite are **Enable CA Password/Disable CA Password**.

If the CA password is not specified or is incorrect, then **Enable CA Password** is shown. Specify and verify the CA password before saving it. If the attempt to save a password exceeds three times, then a warning dialog box opens.

When a user accesses the **Certificate Manager**, and has read permission for the `passwords` node, the CA password is read from the database.

When the **Remember CA Password** option is selected on the **Certificate Manager**, the CA password is loaded into the cache. This is used in all places of the **Certificate Manager**; otherwise, the password is read from the file each time.

Advanced security users and roles

Advanced security enables the security administrator to assign not only user-level permissions, but also role-level permissions.

A role is an arbitrary association based on whatever criteria the administrator selects. For example, common tasks that are performed by multiple users, or the requirement of certain users for certain types of information.

Role-level permissions offer the convenience of setting permissions for any number of users at once. By combining user-level permissions and role-level permissions, the administrator can implement a complete security structure with whatever degree of compartmentalization is required.

Permissions are assigned to roles in the same way they are assigned to individual users. For example, there may be several quality assurance specialists who must perform the same tasks to test and analyze system performance.

Instead of setting the same permissions for each of those users, the administrator can:

- Create a role called Quality Tester.
- Set permissions once for the Quality Tester role.
- Make all those users members of that role.

Roles can also be members of other roles. For example, the Quality Tester role might be a member of the System Operator role. This automatically extends all permissions that are granted to user members of the System Operator role to all user members of the Quality Tester role.

Interaction of user- and role-level permissions

Sometimes, you must individualize permissions for a given user. This is even when that user has a basic function similar to that of other users. Advanced security is designed so that role-level permissions and user-level permissions can be combined to define the appropriate level of access for each individual user.

User membership in multiple roles

Any user can be a member of as many roles as necessary and thereby acquire all the permissions of all those roles.

For roles and permissions, user-level permissions always take precedence over role-level permissions. To ensure a user is granted or denied a specific permission, you can grant or deny that permission at the user level.

Roles as members of other roles

Making a role a member of another role extends all permissions that are granted to the second role to all members of the first role. If a role belongs to two or more other roles, then the member role is granted all permissions that are granted to the roles to which it belongs.

Permissions that are explicitly specified for the lower-level role take precedence over those specified for the higher-level role. For example, suppose the System Operator role was denied Insert permission for a resource, but the Quality Tester role was granted that permission. In that case, the members of the Quality Tester role would be granted the Insert permission.

Adding a role

When you first start building your ACLs, begin by adding names and remarks for all of the roles that you know about. Do not attempt to identify any roles to which the new roles belong, or any roles that belong to the new roles.

Then, you can return to the roles added and modify the original information to create the role-in-role membership structure. The main reason for this is that you can only specify membership for roles that have previously been added.

It is best to add and organize all roles before adding any users. Then the role or roles for each user can be specified as that user is added.

After the first iteration, include membership information about a role when it is added. This depends upon whether the roles and users that you plan to identify already exist in your ACL structure.

- 1 Click the **Roles** tab.
- 2 Click **Add New Role** to open the **Add/Modify Role** dialog box.
- 3 In the **Role Name** field, specify a unique name for the new role.
- 4 Add any comments about the role in the **Remark** field.
- 5 To make the new role a member of one or more existing roles, click **Roles**. Then select the appropriate roles from the resulting **Select Roles** dialog box. Click **OK**.
- 6 To identify one or more existing users or roles as members of the new role, click **Users/Roles**. Then select from the resulting **Select Users/Roles** dialog box. Click **OK**.

Modifying a role

When you modify an existing role, the **Add/Modify Role** dialog box is displayed with the current information about that role. It contains all the information that exists at this point, which might be only the name.

To change a role name, edit **Role Name**. When you click **OK**, the old role is deleted and the new one is added.

Member Of lists all the roles to which this role belongs. Click **Roles**. Then, select the roles to be added/removed from the ensuing **Select Roles** dialog box. This dialog box lists all the available roles, showing which ones to which the role already belongs.

When adding to or removing from existing roles, hold down the **Control** key when you click the roles that you must add or remove. Otherwise any role you click becomes the sole selection.

Members lists all the users and roles that belong to this role. To edit, click **Users/Roles**. Then, select the users or roles to be added or removed from the ensuing **Select Users/Roles** dialog box. This dialog box lists all the available roles and users and shows which ones already belong to this role.

Making a user into a role member

Use ACL/Role Manager to add users only in exceptional circumstances. For example, when your organization upgrades from basic to advanced security or when reinstating a user whose certificate has been revoked.

In normal circumstances, use Certificate Manager to add users by issuing their user certificates.

You cannot issue a user certificate to a user who has been added with the ACL/Role Manager.

When building your ACLs, it is usually best to set up your complete role structure before identifying the users who belong to roles. After the role structure is in place, identify user members in one of these ways:

- Modify each role to add the users that belong to it. If one role belongs to another, then adding a user to the member role automatically adds that user to the other role.
- Modify each user to specify which roles to which that user belongs.

To identify role membership by modifying a user:

- 1 In the ACL/Role Manager, click the **Users** tab and select the user to be modified.
- 2 Click **Modify User**.
- 3 Add comments about the user in **Remark**.
- 4 Click **Roles**.
- 5 Select the roles for this user.

If one role belongs to another, then adding the user to the member role automatically makes that user a member of both roles.

Importing users and groups into ACL Role Manager

The ACL Role Manager gets LDAP user and group configurations through the security server.

The security server retrieves user/group information from the LDAP server according to its LDAP configuration in advanced security. A search filter is constructed internally based on the LDAP configuration.

When ACL Role Manager requests a user list or group list, the security server sends it back.

- 1 Configure LDAP on the LDAP tab of the **Server Administration > Security Server** tab.
 - a Specify a host name.
 - b Use the defaults for **Encryption Method** and **Authentication Method**.
 - c Specify the **Default Domain Name**.
 - d Specify the **Manager Distinguished Name**.

The **Manager Distinguished Name** must be in CN=XXX,OU=XXX,DC=XXXX format.

Depending on how you define your LDAP structure, CN means common name, OU means organization name, and DC means domain component.

There is no length limit.
 - e Specify the **Manager Password** and confirm it.
 - f In the Synchronization section, use the defaults for **Mode** and **Cron**.
- 2 Click **Advanced** and configure classes and search bases for user, group, and server.

Ensure the class and search bases are correct for each type; otherwise, the search result could be empty.
- 3 Click **OK** on the **LDAP Advanced Configuration** dialog box.
- 4 Click **Test** in **Server Administration**.

Verify that **Query User Result** and **Query Group Result** are not empty. They should show the search base and search filter for user and group, respectively.

- 5 Click **Save**, and restart the security server. The security server must be restarted after making LDAP configuration changes.
- 6 Open the ACL Role Manager. In this dialog box, you can import user and groups.

Advanced security permissions

Use advanced security to specify which nodes a user can access and what the user can do at each node. For example, you can specify that certain users can read the configuration files on a site, but not change them. You can also specify that certain users can add new configuration files, but not delete existing ones.

This table shows the different types of permissions that a user can be granted, or denied, for each node:

Permission	Description
Read (R)	Permission to read from the resource.
Write (W)	Permission to write to or edit the resource.
Delete (D)	Permission to delete the resource.
Insert (I)	Permission to insert or create a new resource of this type.
Execute (X)	Permission to run the resource.

Defining Access Control Lists (ACLs)

ACLs are organized in a tree:

- The root tree node is called `host`.
- The second level tree node is the `host` name of the machine where the host server is installed.
- The third level tree node is the Cloverleaf root. For example, `integrator6.2`.

These nodes are under the `Cloverleaf` root node:

- `application`: This is defined in the root level.
- `command`: This is defined in the root level.
- `config`: This is defined in the root level.
- `webservices`: This is defined in the root level.
- **Individual site**: ACLs in this level are defined in the individual site.

`application`, `command`, `config`, and `webservices` define the ACLs in the root level. The ACLs in the site level are defined in the individual site.

The ACL tree node structure is:

```
- host
- - host01
- - - integrator6.1
- - - - application
```



```

- - - - command
- - - - config
- - - - webservices
- - - - site01
- - - - - application
- - - - - command
- - - - - config
- - - - site02
- - - - - application
- - - - - command
- - - - - config
- - - integrator6.2
- - - - application
- - - - command
- - - - config
- - - - webservices
- - - - site11
- - - - - application
- - - - - command
- - - - - config
- - - - site12
- - - - - application
- - - - - command
- - - - - config

```

You can define the ACLs for these tools or applications in the `application` root level tree node:

- `clapi`: The restful API published on host server
- `hciaclrolemgr`: **ACL/Role Manager**
- `hciauditlog`: **Audit Log Viewer**
- `hcibox`: **Box Manager**
- `hcicertmgr`: **Certificate Manager**
- `hciguisiteini`: **Site Init GUI**
- `hciserveradmin`: **Server Administration**
- `usercmd`: This is under `command` and are user commands which are not listed in the IDE commands.

You can define the ACLs for the file operation in the `config` root level:

- `box`: `box`
- `eo`: Engine Output Alias
- `json`: JSON
- `proc`: TCL proc
- `rootInfo`: `rootInfo`
- `userfile`: upload file

You can define the ACLs for the web service in the `webservices` root level.

The nodes under `site` represent the resources of the site level. `application` is for applications such as:

- `hciaccess`: IDE
- `hcialertconfig`: **Alert Configuration**
- `hcinetconfig`: **Network Configurator**
- `hcinetmonitor`: **Network Monitor**

The `command` nodes are commands for actions. For example, `hcicmd` for Network Monitor and `config` for the configuration files, including `alert`, `frl`, `hl7`, and `NetConfig`.

These are the permission types for user and roles:

- execute - e (for application and command)
- read - r (for config)
- write - w (for config)
- insert - i (for config)
- delete - d (for config)

In most cases, if you have permission of `config`, you must run permissions for the specific application. Then, you must read, write, insert, and delete permissions for the specific `config`.

Some configurators require additional run permission for specific commands, for example, `NetConfig` and `NetMonitor`.

Example

This table shows how to configure the permissions using a host server named `CNSHN01` and a current version of 6.2:

Description	Resources	Permissions
Full access to all	host	r, w, i, d, e
Read access to site hell oworld	host/chshn01/integrator6.2/helloworld	r
Full access to site produ ct	host/chshn01/integrator6.2/product	r, w, i, d, e
Deploy BOX	host/chshn01/integrator6.2/application/hcibox host/chshn01/integrator6.2/config/box	e i
Alert Configurator On site helloworld	host/chshn01/integrator6.2/helloworld/application/hcialer tconfighost/chshn01/integrator6.2/helloworld/config/alert	e r, w, i, d
Translation Configura- tor On site helloworld	host/chshn01/integrator6.2/helloworld/application/ hcitranslateconfig host/chshn01/integrator6.2/helloworld/ config/xlate	e r, w, i, d
NetConfig On site helloworld	host/chshn01/integrator6.2/helloworld/application/hcinet onfig host/chshn01/integrator6.2/helloworld/conf ig/NetConfig host/chshn01/integrator6.2/helloworld/config /multiview host/chshn01/integrator6.2/helloworld /config/siteInfo	e r, w r, w r, w
Start NetMonitor	host/chshn01/integrator6.2/helloworld/application/hcinetm onitor host/chshn01/ integrator6.2/hell oworld/command/hcisitectl host/chshn01/ integrat or6.2/helloworld/config/multiview host/chshn01/ integrator6.2/helloworld/config/NetConfig	e e r r

ACL template

The ACL template defines the access level of user action.

The name that is used in the action `refACL` (reference ACL) is required and must be unique. This name cannot contain any of these special characters:

| / \ [] : ; | = , + * < > @ "

To configure:

- `host`: This is the host name. If this is not specified, then all hosts are included.
- `version`: This is the Cloverleaf version, for example, 19.1. If this is not specified, then all versions are included.
- `site`: This is the site name. If this is not specified, then all sites are included.
- `masterSiteOnly`. The default is `false`. If `true`, then do not specify `site`.

For example, to grant lookup table permission on sites named `helloworld` for all host servers whose version is 19.1, you can use:

```
<acl name="helloworld" versions="19.1" sites="helloworld">
  <user name="sample_user">
    <action tool="LookupTable" refACL="helloworld" />
  </user>
```

Roles and users

To configure roles and users:

- `name`: This is required and must be unique.
- `roles`: This can be more than one role, separated by a comma.
- `remark`: This is an added description.

For actions:

- `deleted`: The default is `false`. When the level and tool name are the same, actions, or one action, can be deleted, as it is created/updated by default.
- `rights`:
 - `P`: This indicates "grant" (default).
 - `N`: This indicates "deny".
- `actions`: Actions are separated by a comma.
This relates to buttons and actions in the tool. For example, `open` opens a tool, `start` starts a thread, and `stop` stops the thread.
- `tool`: The tool name in the GUI. This is case-insensitive, for example, `Netmonitor`.
- `refACL`: The reference ID of the ACL.

Setting permissions

When setting permissions, first define your initial users and roles. Then, construct your ACLs by setting permissions for the appropriate users and roles at the various nodes.

- 1 In the ACL/Role Manager, click the **ACLs** tab.
- 2 Click a node to set permissions.

This shows the current permission settings for the node in the right pane of the **ACLs** tab.
- 3 Click **Add**.

This opens the **Permission Modification** dialog box.
- 4 Click **Users/Roles**.

This opens the **Select Users/Roles** dialog box, which lists all the available users and roles.
- 5 To select a user or role, click the listing.

Select more than one user or role to assign them the same permissions.
- 6 Click **OK**.

This closes the **Select Users/Roles** dialog box, and returns you to the **Permission Modification** dialog box. The users or roles that you selected are now listed in **Users/Roles**.
- 7 Specify the permissions to be granted or denied at this node by clicking the appropriate options.

The run (execute) right is not available when you add or modify permissions on the node that delegates a configuration or a configuration set. For example, the config node under a site node or the netconfig node under config.

Granting or denying a permission supersedes the setting at a higher-level node. Leave a permission as "Not Specified" to apply the setting that is specified at the next higher-level node. The applicable setting is determined by the ACL at the node closest in the hierarchy to the selected node.

For example, a user is granted run permission at the node for a system root but denied it at the child node for a site. Leaving the run permission unspecified at `hcicmd` indicates that the user cannot run any of the commands that `hcicmd` contains. That same user could be explicitly granted run permission at the nodes for particular commands.

Any permission that is not specified throughout the hierarchy is granted.
- 8 Click **OK**.

This returns you to the **ACLs** tab, which now shows the newly added permissions. These permission settings apply to this node and all the nodes that it contains.

Read-only permission

When you access a file, if you have no write permission, but read-only, a message box opens. This states that insufficient write privilege exists on that file and it can only be opened as read-only.

After that, the read-only file open status is flagged on the left of the status bar of the IDE.

When you switch the tool tab or changing the content of the current tool tab, the file open status on the status bar is updated.

Network Configurator

The read-only support in Network Configurator is different because the NetConfig and multiple view files are open simultaneously.

- In some instances, you might not have write, but only read permission, on both the `NetConfig` and multiple view files. When this happens, a message opens and "Read-Only" is shown on the status bar.
- If you have no write, but only read permission on the `NetConfig` file or the multiple view file, then a message box opens. This states insufficient privilege on the resource and opens it as read-only. The status bar shows "Read-Only NetConfig" or "Read-Only view."

Other scenarios

When a file is opened as read-only when the file lock conflicts, the read-only file status is at the left of the status bar.

- If you try to modify and save a file opened as read-only, then a message reminds you that the file is read-only.
- In some instances, you might not have read, but do have write permission on that file. You can still read or write it, as if you have read and write permission.

Reviewing permission settings

- 1 In the ACL/Role Manager, click the **ACLs** tab.
- 2 Click the node to review its permissions.
- 3 Right-click anywhere in the left pane to open the context menu and select **Permissions Check**.
This opens the **Permissions Check** dialog box. This dialog box shows on which node the permissions are defined. It also shows the user names that are authorized to access your system, and the specific permission settings for each user at the selected node.
- 4 When you have finished reviewing permissions, click **OK** to close the dialog box and return to the **ACLs** tab.

Modifying permission settings

Any changes that you make to the permission settings for a container node also apply automatically to all of its content nodes.

- 1 Click the **ACLs** tab.
- 2 Click a node to modify its permissions, and then select a user name.
- 3 Click **Modify**. This opens the **Permission Modification** dialog box, showing a list of the current permissions for that user or role.
- 4 Make necessary changes to the current permission settings by clicking the appropriate options.
- 5 Click **OK**.
This closes the **Permission Modification** dialog box. The listing in the right pane of the ACL/Role Manager shows the new permission settings.

Deleting permission settings

Deleting the permission settings at a container node also deletes them at all of that node's content nodes. The deleted user or role cannot perform any operations at any of those nodes.

- 1 From the **ACLs** tab, click the node where you must delete permission settings.
This populates the dialog box with a list of users and roles that have permissions set for that node. The listing for each user or role shows the current permission settings.
- 2 Select the user whose permission settings you must delete.
- 3 Click **Delete**.
This opens the **Confirm Delete** dialog box, which prompts you to confirm your intention.
- 4 Click **Yes**.
This closes the **Confirm Delete** dialog box, and returns you to the **ACLs** tab. The right pane no longer lists the user whose permission settings you deleted.

Multiview

In earlier versions, Network Configurator used a single view file, and Network Monitor used a multiview file. In later versions, both NetConfig and Network Monitor use a multiview file to save view data. This is saved as a single view and multiview combined. The view element is still listed on the ACL configuration to be backward compatible.

The `view` directory contains both single and multiple view files. Although they are all view files, they are controlled by different ACL elements.

For the single view, assign the permission on the view element. For multiple view files, assign the permission on the multiview element. For example, if a user must open a multiple view file, you must grant the permission on the multiview element.

Security Server Migration tool

To run the Security Server Migration tool:

- 1 Run `ssmigration.bat`. This is located at `%HCIROOT%\clgui\bin\ssmigration.bat`.

The version is 2022.09.00.

Usage:

```
ssmigration [-srcVersion 19.1|20.1 -srcRoot source root directory [-logfile log file full path]]  
[-h | -help]
```

- `-srcVersion 19.1|20.1` is the source product version number.
Use 20.1 if there is no `-srcVersion` option.
- `-srcRoot source root directory` is the source product root full directory.

Use "c:\cloverleaf\cis20.1\integrator" if there is no `-srcRoot` option.

- `-logfile log file full path` is the full path file name of the log file.

Use "%HCIR00T/ssmigration.log" if there is no `-logfile` option.

- `-h` or `-help` displays this help message.

- 2 Running this command opens the Migration Wizard at the welcome page (%HCIR00T/security/ssmigration/migrategui.bat).

Note: Before beginning the Wizard, ensure that both CIS versions are shut down.

Click **Next** to open a dialog box for specifying the Security Server location. Select a version and specify, or browse to, the install location.

Click **Next** to open the **Migration Ready** page.

Click **Finish** to start the migration.

Click **Done** to quit the Migration Wizard.

Security log in

Credentials are the files used to access the system with the security server.

You must complete the credentials licensing process before installing or upgrading to security server.

User credentials

Security server licensing involves the creation of user credentials, a private key and certificate.

- Customer CA , Certificate Authority, files are created by Infor for the security administrator of an organization. The Customer CA credentials authorize the security administrator to issue User certificates, and to access the system with security server.
- The user credentials that are created by the security administrator authorize individual users to access and operate the system with security server.

Preferred licensing method

The issuer of the credentials creates an information file and sends it to the user. In this procedure, each user creates a private key, a public key, and a certificate request file. The user then sends a User Information file containing the certificate request back to the issuer.

After checking the contents of the certificate request file, the issuer sends a certificate to the user. The user then tests the certificate to ensure it matches the private key.

There are two advantages to this procedure:

- The user's private key is never exposed. Users can prove that no other person has seen the private key.
- The certificate request and certificate can be transmitted "in the clear" because they contain only public information.

Alternate licensing method

The security administrator also has the alternative of creating and distributing user credentials without individual user participation. This second method eliminates the requirement for these users to master the procedures to creating private keys, public keys, and the certificate request. The CA is responsible for all keys.

PEM format

All of the files that are used in these procedures are encoded in a binary format called ASN.1-DER. Before the time of email attachments, binary files had to be converted to printable ASCII to send them through SMTP servers. The PEM format for any file is an ASCII representation of the ASN.1-DER format.

This is provided as a convenience for users that cannot accept email attachments. Binary files can be sent as attachments or the PEM format can be cut and pasted into an email.

When the security administrator pastes this into the **New Credentials** dialog box, the certificate is saved as a .der file in the correct directory.

Although the PEM format is convenient, the preferred method is to send the ASN.1-DER file as a binary attachment. This method makes the file less prone to error.

Roles

In the licensing procedure, Infor takes the role of certificate authority. The Security administrator of your organization is the customer CA. This is to provide the customer CA with a valid private key, public key, and a customer CA certificate that is digitally signed by Infor.

The customer CA certificate, which is both a User certificate and a CA certificate, makes the security administrator the certificate authority for your organization. The security administrator can then issue certificates to any user. This is to provide users with a valid private key, public key, and a user certificate that is digitally signed by the CA.

End users cannot use their certificates to act as CAs (Certificate Authorities) and create other certificates. The customer certificate has an attribute where it can be used as a CA. The user certificate lacks this attribute.

Obtaining a user certificate

Private key and certificate credentials are required to access a system with security.

Your organization's security administrator has already begun the credentials process. Now, the user to receive those credentials joins the final steps.

In this section, the certificate authority is the security administrator, which is called the customer CA.

By this time, the security administrator has created a User Information File for each user to be given access to the system running security.

Each user receives an email with an attached file or the PEM version. This is pasted onto the emailed message. This attachment, or pasted text, is the user information file.

Certificate Manager

The Certificate Manager is a major component of the security server. The Certificate Manager enables the security administrator to issue/revoke user certificates. These certificates authenticate all the individuals who are authorized to access the system.

During security server setup, the Certificate Manager is installed on one and only one of the computers that run host server. This should be the security administrator's computer. Having the Certificate Manager on one computer not only simplifies certificate administration, but also helps prevent unauthorized access to security-related functions.

Note: Beyond basic security, security server offers advanced security. This gives the security administrator full control over who accesses the system and what each user can do with each function. The system must be authenticated. Users must be issued certificates before they can be added to the ACL (Access Control List). Advanced security uses this list to determine which features and functions can be accessed by individual users.

These topics are intended for the security administrator:

- [Certification](#)
- [Starting Certificate Manager](#)
- [Exiting Certificate Manager](#)
- [Issuing user certificates](#)
- [Issuing host server certificates](#)
- [Issuing security server certificates](#)
- [Viewing certificate information](#)
- [Changing a user password](#)
- [Revoking and un-revoking certificates](#)
- [Refreshing the CRL](#)

This topic is intended for the user:

[Requesting a user certificate](#)

Service Setting dialog box

Selecting **Options > Service Setting** opens the **Service Setting** dialog box. This contains settings for the Notification and Confidential policy.

- **Enable pending request notification**

When this is selected, the administrator receives an email notification that there are requests pending.

- **Enable certificate expiration notification**

When this is selected, the administrator receives an email notification regarding which user's certificate is about to expire on what date. The default value of **Before** is 10 days.

- **Administrator Email**

This is enabled when of the notification options are selected. This email is used for the notifications.

- **Enable case insensitivity for user certificate authentication**

After the host server is upgraded to security mode, a log-in dialog box opens when connecting to the host server.

When you specify the user name, different cases are treated as different users. For example, **administrator** and **ADMINISTRATOR** are identified as two different users. If this is not required, then you can select this option. When this is selected, both specified user names, regardless of case, are considered the same user.

When a notification option is selected, notifications are kept in `notification.log`. This is located at `$HCIRoot\server\logs`. This log is used to record the pending notification and expiration notification for the user certification request.

At times, `notification.log` can grow large and take up disk space. In this situation, you can turn off the log by clearing the notification check boxes. It can also be manually deleted.

Certification

Certification is the first step in securing a system. You can use the Certificate Manager to create certificates that authorize user entry. Each user must have a certificate.

Each certificate consists of these related files:

- Public certificate file
- Private key file

Note: To obtain these files, go to the Infor Support Portal or Concierge and create a case to have the certificate files generated for you. Provide the name in which the certificates must be generated. A security upgrade can be completed only if these two files have been copied to your computer.

The private key file is accessed only by specifying a password, which must be issued to the user that is identified by the certificate.

Certification begins with a CA or Certificate Authority. This is the entity that issues certificates and vouches for the information they contain. Infor is the ultimate CA for the system, which in turn makes your organization a CA when you install advanced security.

To make your organization a CA, Infor gives it a unique CA public certificate file and a unique CA private key file. A CA password is also given that must be specified during Advanced Security setup.

As the security administrator, you require the CA password so that you can use the CA private key file to access the Certificate Manager.

The security administrator must ensure that the correct files are copied to the correct clients, and that the correct user receives the correct password.

Note: The effectiveness of certification depends on the security of the underlying operating system. Each user must have a unique log-in for any machine that runs a client. Any user who has command-line access to a Host Server must also have a unique log-in for that host machine. If users are permitted to share log-ins at the operating system level, then they can impersonate each other and subvert certification.

Selecting **Options > Edit Default Certificate Info** opens the **Edit Default Certificate Info** dialog box, which is similar to the **Issue Certificate** dialog box. This information is used to pre-populate the user information and expiration date in the certificate issuing process.

Securing the user certification process

Certificates are one of the main components of advanced security. Because the certification process itself must be secure, several aspects of the Certificate Manager contribute to secure certification.

Only a CA who logs on with the CA password can issue and revoke user certificates.

Each user certificate consists of two files, a public certificate file and a corresponding private key file. These must be decrypted with the user's password before the user can gain access to the system. The file set for each user certificate is unique.

User certificates must be created on the computer that runs the Certificate Manager. They cannot be created remotely.

User and public certificate files are stored in an encrypted form.

Private key files can be created on the computer that runs the Certificate Manager or on end-user computers.

- If private key files are created on the computer that runs the Certificate Manager, then public certificate files and private key files must be manually distributed. These go to all the computers that run clients.
- If private key files are created on end-user computers, then they can be incorporated into Certificate Requests. These can be encrypted and emailed to the Security Administrator and then decrypted and used to generate User certificates. These can then be emailed to end-users.

Certificates can be revoked by the security administrator. A CRL (Certificate Revocation List) is stored on the computer that runs the Certificate Manager, and checked whenever a user attempts to log in. A user whose certificate is on the CRL cannot complete the log-in.

If a CRL expires, then no one can log in to any client without the CA password. Only the security administrator can log in until the CRL is refreshed.

Certificate states

The **User** tab contain different types of certificate files. Each tab lists certificate files by user name, and shows when each certificate expires.

- **Pending**

This lists all pending user registration requests.

For Action, selecting **Approve the user request** opens a dialog box from which the Administrator generates the user certificate. On this dialog box, when **Send certificate notification email to the user** is selected, an email is sent to the user notifying that the registration has been approved.

Selecting **Deny the user request** marks the user as denied and sends an email to notify the user.

- **Approved**

This lists all approved users. Two actions are available:

- Renewing the expiration date of the certificate
- Revoking the user certificate

- **Revoked**

This lists all users with revoked certificates.

- Un-revoking removes the certificate from the CRL.
- Deleting removes the user request, private key, and certificate from the server side.

- **Denied**

This contains a list of all denied users.

- Reversing adds the denied users back to the Pending list.
- Deleting removes the user request from the server side.

Passwords and the user

When the end-user's private key is created, a password must be created to enable the end-user to unlock the private key. The user can change this password at any time. The original password is saved and remains valid, so the original password can be reissued to any user who forgets the current one.

During log-in, a user has the option of instructing the system to cache credentials. This saves the user's log-in information for reuse if the user accesses another dialog box. The same user can log in one time, and then run multiple dialog boxes without having to log in again.

Password options include:

- **Remember CA Password**

This automatically fills in your Customer CA password when you perform certificate-related actions such as issuing or revoking certificates.

If this option is not selected, then you must specify the CA password every time you sign or revoke a User certificate.

- **Use Default Password**

Select this to use the default password when generating a private key when issuing a host server or security server certificate.

The latter applies if the Host Server has been upgraded to advanced security.

If the default password is never set, then you are still asked to specify a password for private key generation.

- **Set Default Password**

This is active when **Use Default Password** is selected. Use this to set the default password. The **New Certificate Default Password** dialog box is displayed where you can specify a new password.

Audit Log

The Audit Log is accessed from the Start menu at **All Programs > Infor Cloverleaf Integration Suite > Tools > Audit Log Viewer**.

The Audit Log is a record of security-related events, which can be reviewed at any time by the security administrator.

Each entry in the list shows:

- Type of event.
- Date and time it took place.
- Host machine where it took place.
- Brief description.

The Audit Log must be manually enabled through the **Server Administration > Monitor** tab. Otherwise, logging does not take place.

Creating the web services certificate

Web services respects the host server security configuration.

- If the host server is configured in "no security" mode, then no security is used for web services.
- If the host server is configured as basic or advanced security, then Transport Security (SSL) is used for web services.

If a security upgrade changes the security mode to basic or advanced security, then it adds the security configuration to web services. It does this by importing the host server's public key certificate and private key certificate into a keystore file, which is required for Transport Security (SSL).

Select **File > Prepare Web Services Certificate** to configure this certificate.

For **Certificate Path** and **Private Key Path**, click **Browse** to specify the directory path to the security certificates.

Certificate expiration

The **Renew** option is used to re-issue a user certificate when it expires. This uses the existing user certificate, changing only the expiration date. For versions earlier than 6.0, you must re-issue the user certificate as the first time it is issued.

Note: The Certificate Manager emails an alert to the administrator when a certificate is about to expire.

Updating the CA certificate

Send your existing CA certificate to Support to have the expiration date updated.

After the CA certificate is updated, Support sends back the updated CA certificate along with the hie certificate.

Apply both certificates to all of your system and Global Monitor environments.

All existing user, host server, and security server certificates continue to work as before, if not expired.

Updating or renewing issued certificates

- 1 In the Certificate Manager GUI, double-click **Issued Certificates** and select the certificate to be renewed.
- 2 Right-click the certificate and select **Renew**. You can also select **File > Renew**. The **Renew Certificate** dialog box is displayed.
- 3 Leave the **Expire Date** and **Days** fields blank, or specify a date.
 - If left blank, then the default new expiration date is 20 years from the date of issuance.
 - If specifying a new date or days, then the expiration date must be within the range of the current date and the CA certificate's expiration date.
- 4 Click **OK**.

Starting the Certificate Manager

Note: Before you start the Certificate Manager, you must have a certificate and a private key of your own. For this process, see Obtaining user credentials.

You must be on the machine that runs the Host Server to start the Certificate Manager.

This information is required:

- The user name and password that are assigned to the security administrator.
Initially, the name is set to "administrator" by Infor. The password is set during the system installation process.
- The CA password that is assigned to your organization by Infor.

Accessing and logging on to the Certificate Manager

From the IDE, select the Shell Window tool. At the command prompt, specify **hccertmgr**.

From the Windows Programs menu, select **Start > Programs > Infor Cloverleaf Integration Suite > Tools > Certificate Manager**.

At the Command Prompt (DOS or UNIX), change directories to the `/clgui/bin` directory for your system root and specify `hcicertmgr`.

After starting the Certificate Manager, you must log in. Click **Advanced** to expand the dialog box to show the **Certificate Path** and **Private Key Path** fields.

Click the button next to the **Password** field to change the password.

Click **New User** to open a dialog box for new user registration.

Adding new users

- 1 If you are a new user, then click **New User** on the **Please Login** dialog box to begin registration. The **User Registration** dialog box is displayed.
- 2 Click the **New User** tab to begin the process.
- 3 After specifying **Name**, **Email**, and **Password**, click **OK** to send the request to the host server.
The certificate request and private key are automatically generated. When a user registers, the certificate request is automatically sent to the server.

The private key is stored in `HCIRoot/client/certs`.

When a user logs on, if the certificate does not exist on the client machine, the certificate with the corresponding private key is automatically downloaded. This is downloaded from the server side to set up the SSL RMI connection.

Logging on

The **Please Login** dialog box opens if you are not already logged on to the system as administrator, or if your log-in credentials are not cached.

Note: The **User Name**, **Certificate Path**, and **Private Key Path**

- 1 Click **Advanced** to show the **Certificate Path** and **Private Key Path** fields.
- 2 Ensure that the **User Name** fields automatically store the settings from the most recent log-in operation. **Change these settings as required.** field contains `administrator`. This is the user name that is defined for the security administrator during system installation.
- 3 Ensure that the **Certificate Path** fields automatically store the field identifies the appropriate certificate file. This file is always in the `%user folder%/.clide/certs` subdirectory of the directory where the system is installed, and it always has the file name `administrator-cert.der`.
- 4 Ensure that **Private Key Path** identifies the appropriate certificate key file. This file is also in the `%user folder%/.clide/certs` subdirectory of the directory where the system is installed, and it has the file name `enc-administrator-key.der`.
- 5 Specify the password for the administrator user ID in the **Password** field.
The button next to the **Password** field opens a dialog box for changing the password.

If you plan to use another dialog box, or if you plan to use the Certificate Manager more than once, then select **Use Credential Cache**. This automatically applies the log-in information throughout the current session.

- 6 Click **Advanced** to expand the dialog box to show the **Certificate Path** and **Private Key Path** fields.

Note: If **Use Credential Cache** is not selected, then you must log in manually each time you start a system dialog box.

- 7 Click **Apply**. This loads the Certificate Manager.

- 8 Click **Advanced** to expand the dialog box to show the **Certificate Path** and **Private Key Path** fields.

- 9 Click **Apply** on the **Login** dialog box. This performs these tasks:

- Checks the certificate and private key in the client.
- Checks that the certificate on the server side is up to date. If it is, then the new certificate and the corresponding private key, if it exists on the server, are downloaded from the server to the client.
- Checks if the private key can be uploaded to the server side.
If it can, then the private key and the corresponding certificate are uploaded from the client to the server.
- Uploads or downloads the certificate and private key based on the above checks.
- Finishes the log-in when the certificate and private key are ready.

Exiting the Certificate Manager

Exit the Certificate Manager in one of these ways:

- Select **File > Exit** to end the session. When security is enabled, you can start any system dialog box without logging back on if these conditions are met:
 - The **Use Credential Cache** check box was selected on the **Please Login** dialog box.
 - The current session has not timed out.
 - The current host server has not been restarted.
- Select **File > log out & Exit** to end the session and log out. You must log back on before starting any system dialog box.
Note: This option is available only when security is enabled.

Issuing user certificates

The Security Administrator is responsible to create and distribute credentials to your organization's end-users.

User credentials consist of two files:

- Private key file
- User certificate file

When a user certificate is issued, the user name is submitted and stored in security server.

You can create credentials for an end-user with one of these methods:

- Create both files for the user.
- Issue the certificates with the user's participation.

Issuing user certificates with user participation

- 1** In the **Certificate Manager** dialog box, select **File > Issue User Certificates**. This opens the **Manage User Certificate Requests** dialog box.
- 2** In the **Add User** field, specify the new user's name.
- 3** Click **Add**.
To batch-process new users, repeat steps 2 and 3 to add as many new users as required before continuing.
- 4** Select the user name in **Select User**.
- 5** Click **Next**.
This reconfigures the dialog box for specifying user information. **User Name** shows the name of the user selected in Step 4 and is not editable. To change the user name, click **Prev** to return to the previous dialog box.
- 6** Fill in the necessary information to create a user information file.
The only required information items are the user name and email address. Everything else is optional.
 - **Country** is the two-letter code. For example, US for United States.
 - **State**, or province, or other governmental unit, is not abbreviated.
 - **Locality** is usually the city name.
 - **Organization** is the legal name of your organization.
 - **Unit** is any unit within the organization, and is user-defined.
 - **Email** is the user's email address.
- 7** Click **Save User Information**. The dialog box displays the location of the information file.
- 8** Email the user information file to the user.
If your email does not support attachments, then send the request in PEM format.
To send the form in PEM format, click **PEM Format** and paste it into an email message.
After receiving the user information file, the end-user selects an algorithm, creates a private key, a public key, and a Certificate Request. At this point the end-user emails you the certificate request.
- 9** Click **Next**. This reconfigures the dialog box for copying the user's Certificate Request to the specified location.
 - If it is an email attachment, then copy it to the location that was specified in Option 1.
 - If it is the text of an email message, then copy the entire message into the text box in Option 2. Copy the message from -----BEGIN CERTIFICATE REQUEST----- through -----END CERTIFICATE REQUEST-----.
- 10** Click **Next**.
- 11** If the end-user did not modify the original user information, then this would be stated in the first item.
If the first item states that the end-user modified information, then review the new information.

If the end-user made unacceptable changes, or if the second item specifies that the end-user did not create a public key, then click **Quit**. Then, notify the end-user to change it and resend the Certificate Request. When you receive the new information, repeat the steps.

- 12** When you are satisfied that the end-user's information is valid, and that the end-user has created a public key, click **Next**.
- 13** Use the **Start Date or Days** field to specify a start date for the user certificate other than the default. The date can be expressed in mm/dd/yyyy format or as a specific number of days from the current date, such as 1000.
- 14** In the **Expire Date or Days** field, specify the expiration date in mm/dd/yyyy format, or the number of days from the current date, such as 1000.
You can also accept the valid date range that is shown in the dialog box and not specify any dates or days.
Note: An expiration date, or number of days, is required. A user certificate can never outlive a CA certificate. If the expiration date is after the CA certificate's expiration date, then the user certificate is automatically set to expire one day before the CA certificate's expiration date.
- 15** Click **Create Certificate**. A message is displayed confirming that the password that was created for the user.
- 16** Specify the password and click **OK**. The Certificate Manager creates the user certificate and notifies you that the certificate has been created.
- 17** Click **Finished** to add another user or **Quit** to exit the GUI.
- 18** Click **Quit** to close the **Manage User Certificate Request** dialog box.
- 19** Verify that the user certificate has been issued by clicking **File > Refresh** in the **Certificate Manager** dialog box.

Issuing user certificates without user participation

The procedure for issuing a user certificate without user participation is identical to the procedure for issuing user certificates with user participation, up to Step 7.

- 1** In the **Certificate Manager** dialog box, select **File > Issue User Certificates**. The **Manage User Certificate Requests** dialog box is displayed.
- 2** In **Add User**, specify the new user's name.
- 3** Click **Add**. This adds the user name to **Select User**.
To batch-process new users, repeat steps 2 and 3 to add as many new users as required before continuing.
- 4** If necessary, then select the user name in **Select User**.
- 5** Click **Next**. This reconfigures the dialog box for specifying user information.
User Name displays the name of the user selected in Step 4 and is not editable. To change the user name, click **Prev** to return to the previous dialog box.
- 6** Fill in the necessary information to create a User Information File.
The only required information items are the user name and email address. Everything else is optional.
 - **Country** is the two-letter code. For example, US for United States.
 - **State**, or province, or other governmental unit, is not abbreviated.

- **Locality** is usually the city name.
 - **Organization** is the legal name of your organization.
 - **Unit** is any unit within the organization, and is user-defined.
 - **Email** is the user's email address.
- 7 Click **Save User Information**.
 - 8 Click **Next**. This reconfigures the **Manage User Certificate Requests** dialog box.
 - 9 Click **Make Private Key**. The **Enter Password** dialog box is displayed.
 - 10 In **Please enter the password**, specify the password to access the new user's private key file.
 - 11 In **Confirm Password**, specify the same password. Make a note of this password! You must send it to the new user.
 - 12 Click **Next**.
 - If the first item states that the end-user modified information, then review the new information.
 - If the end-user made unacceptable changes, or if the second item specifies that the end-user did not create a public key, then click **Quit**. Then, notify the end-user to change it and resend the Certificate Request. When you receive the new information, repeat the steps.
 - 13 Click **Next**.
 - 14 Use **Start Date or Days** to specify a start date for the user certificate other than the default.
The date can be expressed in mm/dd/yyyy format or as a specific number of days from the current date, such as 1000.
 - 15 In **Expire Date or Days**, specify the expiration date in mm/dd/yyyy format, or the number of dates from the current date, such as 1000.
You can accept the valid date range that is shown on the dialog box and not specify any date or days.
Note: An expiration date, or number of days, is required. A user certificate can never outlive a CA certificate. If the expiration date is after the CA certificate's expiration date, then the user certificate is automatically set to expire one day before the CA certificate's expiration date.
 - 16 Click **Create Certificate**. This opens a dialog box for specifying the password that was created for this user.
 - 17 Specify the password and click **OK**.
The Certificate Manager creates the user certificate and notifies you that the certificate has been created.
 - 18 Click **Finished** to add another user or **Quit** to exit the GUI.
This creates two certificate files on the machine with the host server:
 - `username-cert.der`
 - `enc-username-key.der`

username is the user ID that was provided for the certificate.

These two certificate files are placed in the `\server\certs\issued` subdirectory of the directory where the host server is installed.
 - 19 Inform the new user of the password you specified in Step 11.
 - 20 Verify that the user certificate has been issued by clicking **File > Refresh** in the **Certificate Manager** dialog box.

Requesting a user certificate

To use the system with security, you must request user credentials from your organization's security administrator.

The security administrator issues user credentials by creating both files. These are kept on the host server, in the form of a DER file.

There are two methods of requesting a user certificate from your security administrator: From the **New User** tab or from the **New Credentials** tab.

To request a user certificate from the **New User** tab:

- 1 Specify your system user name. This is assigned to you by your security administrator.
- 2 Specify your email address and password. Click **OK**.
- 3 Wait for the approval from your CA.

To request a user certificate from the **New Credentials** tab, follow the instructions on the tab:

- 1 Specify your system user name, which is assigned to you by your security administrator.
 - If you received the user information file as an email attachment, then the user name is also included in the file name. For example, `usernameinfo.der`.
 - If you received the file in PEM format, then the user name is in the first line. This informs you where to save the PEM file. For example, Save the following text as `username-info.pem`.
- 2 Copy the user information file to the specified location.
 - If the file is in DER format, for example, then copy it to the specified location.
 - If the file is in PEM format, for example, then paste it into the PEM text box.
- 3 Click **Use Info File** to copy information from the user information file. This fills in some, or all, of the text boxes, providing the information specified by your security administrator.

Note: You should not change any of this information or complete any blank items, unless you receive explicit instructions from your security administrator.
- 4 Click **Create Private Key and Request** to generate your public key, your private key, and the request for a User Certificate. The **Enter Password** dialog box is displayed.
 - In the **Please enter the password** field, specify the password that is used to generate and access your private key.
 - In the **Confirm Password** field, specify the same password again. The two passwords must match.

Note: Make a note of this password! This is required to test your user certificate when the security administrator returns it.
- 5 Click **OK**. This action:
 - Copies the contents of the User Information File that you completed in Step 3 into a Certificate Request file.
 - Creates a public key and incorporates it into the Certificate Request file.
 - Encodes the Certificate Request file.
 - Creates a private key that only you can use with the password.
 - Opens a dialog box to notify you of the operation's success.
- 6 Email the Certificate Request to the Security Administrator.

The file name and location are specified on the **New Credentials** dialog box. If your email does not support attachments, then send the request in PEM format.

- 7 After you receive the certificate, copy it to the specified location.
 - If the file is in DER format, then copy it to the specified location.
 - If the file is in PEM format, then paste it into the PEM text box. This automatically copies it to the appropriate location and gives it a .pem extension.
- 8 Click **Test** to test the user certificate to ensure it matches your private key.

When you click **Test**, the **Enter Password** dialog box opens.

Specify the password that you specified earlier, when you first opened the **New Credentials** dialog box. This compares the certificate with the private key to ensure that they match.

Note: If the test fails, then someone could have interfered with the procedure. Notify your security administrator immediately to revoke the certificate.
- 9 Click **OK**. This concludes the procedure for obtaining a user certificate and closes the **New Credentials** dialog box.

You can now access the system, logging on with the user name assigned to you by the security administrator and the password that you supplied.

Issuing JKS certificates

The Certificate Manager can generate certificates into a JKS Keystore. The **Issue JKS Certificates** option opens a wizard for generating a JKS store and certificate.

When you specify user information, validation date, and the key and keystore passwords, the Certificate Manager generates a keystore in `$HCIR00T/server/certs/store`.

- 1 Select **File > Issue JKS Certificates** to open the wizard. The first page of the wizard is the **Create a Keystore** page.
- 2 Specify a name and password for the JKS file.
- 3 Click **Finish** to save the JKS file to `$HCIR00T/server/certs/store`. This file is protected by a password and has no content.
- 4 Click **Next** to proceed to the **Create Information of Certificate** page.
- 5 Specify the user information for the certificate.
- 6 Click **Next** to proceed to the "Create Certificate" page.
- 7 Specify the valid date and alias name for the certificate, along with the password of the private key.
- 8 Click **Finish** to create the JKS file which contains a certificate and a private key.

The JKS file is saved in `$HCIR00T/server/certs/store`.

Keystore tab

This tab lists all JKS files which are created by the Certificate Manager.

It displays Keystore as a tree list. The root level is `stores`. The second level contains the JKS files. Clicking a JKS file expands the current node and displays all certificate entities in this JKS file.

Right-clicking opens a menu of options:

- **Add a Keystore**
Adds a JKS file into this stores.
- **Remove a Keystore**
Removes a JKS file from this list.
- **Import a Certificate**
Imports a certificate into a selected keystore.
- **Export a Certificate**
Exports the selected certificate from a keystore.
- **Delete an Alias**
Deletes a certificate or certificate with private key from a keystore.
- **View Certificate**
View detail information of the selected certificate.
- **Refresh**
Refreshes the tree list.

Issuing host server certificates

A host server requires a certificate to authenticate itself in communications with clients. Every host server is issued a certificate during basic security installation. The only time you must issue a new one is if the public certificate file or the private key file is lost or damaged.

1 Select **File > Issue Host Server Certificate**.

Specify this information:

Host Name

Specify the network-resolvable machine name for the computer system where the host server resides.

Strength

Currently, the only possible setting for **Strength** is **512**. This is the size of the public key and the private key that are generated. If the U.S. government regulations change to permit larger keys, then Infor can provide different key sizes. Larger keys are more difficult to decrypt, but increase processing requirements.

Expiration

Specify the date in mm/dd/yyyy format, or the number of days from the current date, such as **1000**.

An expiration date, or number of days, is required. A user certificate can never outlive a CA certificate. Sometimes, an expiration date is specified that falls after the expiration date of the CA certificate. If this happens, then the user certificate that you issue is automatically set to expire one day before the expiration date of the CA certificate.

- 2 Optionally, complete any of the other fields.
- 3 If your organization uses its certificates in communicating with customers, vendors, or other organizations, then specify this information:

Country

Specify the two-letter code. For example, **us** for the United States.

State

Specify the state or province, or other governmental unit. The name is not abbreviated.

Locality

Specify the city name or locality.

Organization

Specify the legal name of your organization.

Unit

Specify any unit within the organization that is user-defined.

Email

Specify the user's email address.

- 4 Click **Issue**.
- 5 Specify this information:

New Password

Specify the password for the host server to use to access its private key file.

Confirm Password

Specify the same password.

- 6 Click **OK**. A certificate is created and a success message is displayed. Default certificate information or a default certificate password is used if they have been provided.

Issuing security server certificates

The security server requires a certificate to authenticate itself in communications with one or more host servers. Every security server is issued a certificate during installation. The only time you must issue a new one is if the public certificate file or the private key file is lost or damaged.

The procedure for issuing a security server certificate is the same as the procedure for issuing a user certificate, with these exceptions:

- For the first step, select **File > Issue Security Server Certificate**.

- The **Issue Certificate** dialog box has **Security Server Host Name** instead of **User Name**. The host name that you specify must be one that your network can resolve.

Viewing certificate information

After a certificate has been issued, you cannot change any of the certificate information. You can view it with this procedure:

- 1 In the **Certificate Manager** dialog box, select one or more certificates for viewing.
- 2 Select **File > View Cert(s)/Change Password**. The **User Information** pane on the right contains the information provided about the user when the certificate was issued.

The Certificate Information pane on the left contains information about the certificate.

File Name

The name of the file that contains the certificate.

User Name

The user name that you specified.

Valid From

The date and time the certificate was issued.

To

The date and time the certificate expire.

Serial No

The unique number that identifies this certificate.

Finger Print

The unique number that is used during certificate authentication in SSL (Secure Sockets Layer) communications.

Previous and **Next** enable you to navigate through multiple certificates.

Password opens the **Modify Password** dialog box. See [Changing a user password](#).

Changing a user password

When you issue a user certificate, you specify a user password for logging on to the system. Then, you copy the public certificate file and the private key file to the machines where that user can log in. After this, inform the user of the password.

Any user can change the original password with a utility called `passwd`, which is located in the `\clgui\bin` directory of each client. This does not affect the password and private key file maintained for that user on the host server. The original password remains valid.

Sometimes a user forgets the new password. You can change the original password and issue a new one.

- 1** In the Certificate Manager, select the user whose password you require to change.
- 2** Select **File > View Cert(s)/Change Password**. The **View Certificate** dialog box is displayed.
- 3** Click **Password**. The **Modify Password** dialog box is displayed.
- 4** Specify this information:
 - Old Password**
Specify the user's original password. This is the password that was specified when the certificate was issued.
 - New Password**
Specify the new password that is assigned to the user.
 - Confirm Password**
Specify the new password again.
- 5** Click **OK**. This closes the **Modify Password** dialog box and returns you to the **View Certificate** dialog box.
Note: You must inform the user of the new password, and ensure that the new private key file is copied to the user's machine.

Revoking and un-revoking certificates

There are many reasons why it might become necessary to revoke a User certificate after it has been issued. Because the user certificate resides on the host machine where the user accesses the client, the Certificate Manager cannot revoke it by eliminating it. Instead, the Certificate Manager relies on a CRL, or Certificate Revocation List.

If the user ID is in the list, then that user is denied access.

The file that contains the CRL is created the first time you revoke a user certificate. This file is called *customer-revoke.crl*. *customer* is the unique customer name that is assigned to your organization by Infor. After the file is created, it is updated whenever you revoke a certificate.

Note: When a user certificate is revoked, the related information is deleted from the security server.

The Certificate Manager creates or updates the CRL file in the `\server\certs\revoked` subdirectory of the directory where the system is installed. To give you the greatest possible control over the CRL, it cannot be used for validation when it is in this subdirectory. This must first be copied one level up, to the `\server\certs` subdirectory. When you are satisfied with the list of revoked certificates, copy *customer-revoke.crl* to use it for access control.

Note: If the `\server\certs` directory does not contain *customer-revoke.crl*, then CRL verification does not happen. All users who have been issued certificates are permitted to access the system.

Whenever a user attempts to log in to the host server, the CRL is checked to see whether it contains that user ID.

If you use multiple host servers, then you must copy `customer-revoke.crl` to every computer that runs a host server. Ensure that it is in the `\server\certs` directory.

Similar to certificates, the CRL has an expiration date. If the CRL expires, then the Certificate Manager has no way to determine which user certificates have been revoked. It automatically denies access to all users except the security administrator.

To help prevent the CRL from expiring accidentally, the Certificate Manager requires you to set an expiration date for the whole CRL. You must do this each time you revoke a user certificate. This reminds you to set the expiration date far enough into the future.

Revoking user certificates

- 1 In the **Certificate Manager** dialog box, on the **Certificate** tab, expand **Issued Certificates**.
- 2 Right-click the certificate to revoke and select **Revoke**. The **Revoke Certificate** dialog box is displayed.
- 3 Click the **Detail** tab to review the selected certificate.

This tab contains two panes with the same information as the **View Certificate** dialog box. It also has a **No Revoke** option that you can use to cancel revocation. If multiple certificates have been selected, then use **Next** and **Previous** to navigate among them.
- 4 When you have finished reviewing the certificate, click the **Main** tab.
- 5 In the **CRL Valid Days or Expiration Date** field, specify the number of days or a date in mm/dd/yyyy format. The expiration date cannot be before tomorrow or after the CA expiration date.
- 6 Click **OK**. The **Revocation Successful** message dialog box displays the successful operation and explains how to enable the revocation list.

This removes the selected certificate from the list in the **Issued Certificates** folder and adds them to the list in the **Revoked Certificates** folder.

Copy and paste the `.crl` file from the `\server\certs\revoked` folder to the `\server\certs` folder.

Note: The **Revoked Certificates** folder lists the date that the certificate would have expired if it had not been revoked.

Un-revoking a user certificate

Un-revoking a revoked user automatically copies the certificates back to the `Issued Certificates` directory and updates the `.crl` file in `/integrator/server/certs`.

- 1 Right-click the issued certificate to open the context menu. Select **Un-revoke** to open the **Un-revoke Certificate** dialog box. You can also select **File > Un-revoke**.
- 2 In the **CRL Valid Days or Expiration Date** field, specify a number or a date in mm/dd/yyyy format.
- 3 Click **OK**. The **Un-Revocation Successful** dialog box displays the successful operation and explains how to update the original revocation list.

This removes the selected certificate from the list in the `Revoked Certificates` folder and adds them to the list in the `Issued Certificates` folder.

Copy and paste the .crl file from the \server\certs\revoked folder to the \server\certs folder.

Refreshing a CRL or an expired CRL

There are two main reasons to refresh the CRL, or Certificate Revocation List:

- If the CRL is about to expire, then you must refresh it to change the expiration date. Otherwise, no one, including yourself, can log in to the host server.
- Sometimes, the Revoked Certificates folder in the **Certificate Manager** dialog box does not match the actual contents of the CRL. If this happens, then you can replace the CRL with a new one that matches the Revoked Certificates folder.

Refreshing the CRL

- 1 In the **Certificate Manager** dialog box, select **File > Refresh CRL**. The **Refresh CRL** dialog box is displayed.
- 2 In the **CRL Valid Days or Expiration Date** field, specify the number of days for the CRL to remain valid, or the date for it to expire. This is in mm/dd/yyyy format. For example, **1** means the CRL expires tomorrow.
- 3 Select the CRL refresh option.
- 4 Click **Update existing CRL** to change the expiration date for the existing CRL.
- 5 Click **Create a new CRL** to use the current contents of the `Revoked Certificates` folder to build a new CRL that contains only those entries.

Be careful when selecting the option to create a new CRL. The \server\certs\revoked folder must contain the certificates of all users who should not be authorized to access the system. If a certificate is not in this folder, then the new CRL does not have an entry for that user.

- 6 Click **OK**.

Refreshing an expired CRL

If your CRL accidentally expires, then no one can log in to the host server.

- 1 In your operating system's command-line interface, change to the \clgui\bin\misc subdirectory of your system root directory.
- 2 Run **hcicrlrefresh**. This prompts for the CA password.
- 3 Specify the CA password. The CA password is not hidden when you specify it, so you should ensure that it cannot be observed.
- 4 Specify the CRL expiration date in mm/dd/yyyy format, or a specific number of days. For example, **1** for tomorrow.
- 5 Specify whether to update the existing CRL or create a new one. The default is to update the existing CRL.

- 6 Specify **1** or press **Enter** to update the existing CRL. The application updates the CRL and reports its success, showing the new expiration date, and then exits automatically.

Audit Log Viewer

The Audit Log is a record of security-related actions, such as successful log-in attempts. This log is accessed through the Audit Log Viewer.

The audit log is kept for all security levels (advanced, basic, or no security). These types of events are logged:

Events	No security	Basic security	Advanced security
Logins	X	X	X
Application run	X	X	X
Configuration file reads/writes	X	X	X
Process run	X	X	X
Issuance/Revocation of certificates		X	X
Updates to the certification revocation list		X	X
Illegal access by unauthorized users		X	X
Updates/Add/Deletes of system users			X
Updates/Add/Deletes of system roles			X
Updates/Add/Deletes of access control lists			X

Note: Security Server (Advanced) is sold separately as an add-on.

For all security levels, the Audit Log Viewer is available on the host server and client and is restricted by user log-in ID.

You must be on the machine that runs the host server to start the Audit Log Viewer.

This information is required:

- The user name and password that are assigned to the certificate administrator. Initially, the name is set to "administrator" by Infor. The password is set during the installation process.

- The unique CA password that is assigned to your organization by Infor.

Enabling the Audit Log

If advanced security is used, then the Audit Log is automatically turned on. If no security or basic security is used, then you must manually turn on the Audit Log.

Enable the Audit Log through the **Monitor** tab of the **Server Administration** GUI. For Windows, go to **Start > All Programs > Infor Cloverleaf Integration Suite > Tools > Server Administration**.

If **Audit Server** is off, then a message is displayed after you specify the `hciauditlog` command or select **Audit Log Viewer** from **Start menu > Tools**.

Accessing the Audit Log Viewer

- 1 Open the Shell Window tool.
- 2 At the command prompt, specify `hciauditlog`.
In Windows, select **Start > All Programs > Infor Cloverleaf Integration Suite > Tools > Audit Log**.
At the command prompt (DOS or UNIX):
 - a Change directories to the `/clgui/bin` directory for your system root.
 - b Specify `hciauditlog`.

Logging on to Audit Log Viewer

After starting up the Audit Log Viewer, you must log in. If you are not already logged on to the system as "administrator," or if your log-in credentials are not cached, then the **Please Login** dialog box opens. Continue with the rest of this section to log in to the system. New users must create credentials. The **User Name**, **Certificate Path**, and **Private Key Path** fields automatically keep the settings from the most recent log-in operation. Change these settings as required.

- 1 The user name is assigned by the administrator, and is defined for the Security administrator during installation.
The Security administrator should log in with the **User Name**, **Certificate Path**, and **Private Key Path** set to "administrator."
- 2 The **Certificate Path** field identifies the appropriate certificate file. This file is always in the `%user folder%/.clide/certs` subdirectory of the directory where the system is installed, and it always has the file name `username-cert.der`.

- 3 The **Private Key Path** field identifies the appropriate certificate key file. This file is also in the %user folder%/ .clide/certs subdirectory of the directory where the system is installed, and it has the file name enc-username-key.der.
- 4 Specify the password that is specified by the administrator for your user ID into the **Password** field.
- 5 If you plan to use more than one GUI, or if you plan to use the same GUI more than once, then select **Use credential cache**. This automatically applies the log-in information throughout the current session. If this check box is not selected, then you must log in manually each time you start a GUI.
- 6 Click **Apply**.
- 7 Click **OK**.

Exiting the Audit Log Viewer

Exit the Audit Log Viewer in one of these ways:

- Click **File > Exit** to start the Audit Log Viewer or any other GUI without logging back on. You must supply the CA password to access the Audit Log Viewer again.
- Click **File > Log Off & Exit** to ensure that no GUI can be started by anyone, including you, without logging on. This option is available only when security is enabled.

Exiting has no effect on any other open GUI, even if you do log out. You can continue to use any GUI that is already open.

Audit Log Viewer dialog box

The **Audit Log Viewer** dialog box lists the security-related events that have happened on one server since the last time the log was cleared.

Each entry in the list shows the:

- Type of event.
- Date and time it took place.
- Host machine where it took place.
- Brief description.

The Audit Log must be manually enabled through the **Monitor** tab of the **Server Administration** dialog box. Otherwise, logging does not take place.

Each log entry reports an event. Each entry has one of these icons that indicates the event type:

- A blue icon indicates an Info or Login Success entry.

An Info entry reports a normal event such as the establishment of a connection between a client and a server, or the retrieval of data from a security-related file.

A Login Success entry reports a successful log-in by any user.

- An blue icon with an arrow indicates a log entry containing logged message content.
- A yellow icon indicates a non-critical Warning entry. A Warning entry reports variations in log-in attempts, such as a privilege denial.
- A red icon indicates an Error entry. An Error entry reports a serious problem, such as an attempt to breach security by an unauthorized user.

The Audit Server records the viewing activities of SMAT messages and records the activities of resent messages.

Sorting log entries

You can sort log table columns by clicking on their headers. Click the column header to put it into ascending or descending order.

The Up or Down icon that is shown inside the column header indicates the order type and the table column that currently determines the sort order.

Initially, the list is sorted by date and time, displaying the most recent event first. You can sort the list in different ways by clicking the appropriate column heading (except for Time, which depends on Date). Any resorting operation works only on the current list. It does not retrieve new entries from the Audit Log.

Audit log search

To aid in searches, the Audit Log toolbar contains a **Search** field and **Search** button.

The search criteria are case insensitive and substring match.

The search function supports cell-by-cell and forward searches in cycle.

When no entry is selected, the Audit Log Viewer searches from the first cell in the first line. If a single entry is selected, then the Audit Log Viewer searches from the next cell of the selected cell in the selected line. When multiple entries are selected, the search begins from the next cell of the beginning selected cell's column in the first selected line.

For example, you can multi-select rows by clicking **Ctrl+cell**. Each cell has its coordinate, that is, (row, column). If you select row 2,4,6 by clicking cell (2,4), (4,1), and (6,5), then it searches from the next cell (2,2) of start cell (2,1). This indicates the start cell has the smallest row number and smallest column number.

The cell that matches the search string is highlighted. If nothing is found, then a message box opens to notify you that the search item was not found. If the text field is empty, then no action is applied when you click **Search**.

Audit log export

This feature is accessed only through the command line. Functionality is exposed through the Cloverleaf API.

For security, you must have a `cl-aom-sec-advanced` key to export the audit log. This tool is licensed with the security server component. Because the audit log is encrypted and can only be viewed through the Audit Log Viewer, the Audit Log Export tool can export the encrypted audit log to a flat-file CSV format. Message content can be optionally exported encoded in base-64.

This tool can export all audit log data, include any message content, and can export audit log entries in a specified date range.

Example: Exporting the audit log

To export the Cloverleaf audit log for the previous day to an external system, write a batch file that calls the `export_audit_log` function with the beginning and end dates equal to the previous day. After this, you can schedule the job to run after midnight, or any other time, using the operating system scheduler.

Example: Exporting the audit log and message data

To export the Cloverleaf audit log and messages from the previous day to an external system, write a batch file that calls the `export_audit_log` function with the beginning and end dates equal to the previous day. You can also write a switch to export the message data encoded in base-64. After this, you can schedule the job to run after midnight, or any other time, using the operating system scheduler.

Audit Log detail panel

A detail panel on the right side of audit log table shows the log message. By default, the detail panel is closed. Clicking **Expand** opens the detail panel.

When a log entry is selected, if the log entry has logged message content, the log detail panel shows content detail.

If the selected log entry does not have logged message content, then the detail panel only shows the log message.

When the log detail panel is open, click the button again to close the panel. When the dialog box is closed, the show/hide state of the log detail panel is saved into the `clide` configuration file. This retains the last state when opened next.

Along with the show/hide state, the size of the viewer and detail panel are also saved into the `clide` configuration file.

See [clide.ini](#).

Detail panel for SMAT messages

If the logged message content of the selected entry is SMAT message data, then the detail panel shows the Message ID, Metadata, and Content of the SMAT message.

When resending, if **Include Metadata** is cleared on the **Resend Message** dialog box, then the logged SMAT message data of the Viewer does not have the Metadata section.

The logged message content can have multiple SMAT messages. This is usually the case when resending SMAT messages. The log detail panel includes a navigation bar at the top of the panel where you can go through the message list.

If the message content does not have multiple SMAT messages, then the navigation bar is not available.

When resending, if **Log message content** is not selected, then the Audit Server records only the message IDs and shows them in the detail panel.

Audit Log preferences

To open the **Audit Log Preferences** dialog box, select **Options > Preferences**. Use this dialog box to customize the operation of the **Audit Log** and **Audit Log Viewer** dialog box.

This table shows the audit log preferences options:

Option	Description
View only the most current entries	Specifies the maximum number of entries that are shown in the Audit Log Viewer dialog box.
Receive Live Audit Log Updates	<p>Determines whether events are automatically listed in the Audit Log Viewer dialog box when they are reported in the Audit Log. New events are added to the list in the position that is appropriate for the currently selected sort.</p> <p>If the list contains the maximum number of entries (see above), then the final (or first) entry is deleted when a new one is added. Deletion order is specified by the user.</p> <p>If this option is selected, then the View > Refresh command is not available.</p>

Option	Description
Log message content	<p>Specifies if the message content should be recorded into the audit log. By default, it is cleared.</p> <p>If it is selected, then the Audit Server records the message content into the audit log file when viewing or resending SMAT messages or outputting the extra-long form of message data through the Database Administrator.</p> <p>If the message content is recorded, then it is encrypted before being written to the log file. The transmission to the client side, viewed on the Audit Log Viewer, is encrypted to ensure safety.</p> <p>When resending, the Audit Server records the message content of the modified messages. For unmodified messages, the Audit Server records only their message IDs.</p> <p>When a message is changed and resent, you can view the message content under any of these conditions:</p> <ul style="list-style-type: none">• Save Outbound Messages is enabled and the message is resent to the thread at the Outbound pre/post TPS queue. See Outbound pane on page 549.• Save Inbound Resent Messages is enabled and the message is resent to the thread at the Inbound pre/post TPS queue. See Inbound pane on page 546. <p>In the Audit Log Viewer, you can view the content when Log message content is enabled.</p> <p>The entry for the resent message is prefaced with "GRANTED execute permission."</p>

Option	Description
Split log entry	<p data-bbox="818 302 1409 464">Specifies whether a log entry should be split into multiple entries before being written to the audit log file. This option is enabled when Log message content is selected, because whether the log entry is split is up to the logged message content size.</p> <p data-bbox="818 478 1409 772">When this option is selected, the kilo-characters box is enabled. This is used to set the maximum size of the message content to be logged onto one log entry. By default, the maximum size is 1000 kilo-characters. A log entry whose message content exceeds the specified maximum size is automatically split into multiple ones. The generated log entries have the same log message and are written to the log file in sequence to ensure they are in a group.</p> <p data-bbox="818 787 1409 949">Each log entry logs part of the message content. The content size for the log entry is determined by the specified maximum size. This indicates that each log entry takes the maximum size to log the message content until the whole message content is logged.</p> <p data-bbox="818 963 1409 1291">When the message content is from a SMAT message, a different way of splitting the message content is used to avoid breaking the integrity of the SMAT message. Each log entry always logs the entire SMAT message. When the size of the SMAT messages in one log entry exceeds the maximum size, the last messages is moved to the next log entry to record until the current size is appropriate. When the size of a SMAT message exceeds the maximum, there is no splitting in one log entry.</p>

Log Size Control pane

The Log Size Control section of the Audit Log Viewer has options for controlling the audit log file size. This prevents the log file from growing indefinitely. By default, **Enable log size control** is selected and the log size control is enabled.

Auto trim log and **Auto cycle log** are enabled when **Enable log size control** is enabled. These are used to specify how the Audit Server handles the log file when the log size control takes effect.

- **Setting Audit Log Preferences > Auto trim log**

When the number of entries exceeds the number in **Maximum number of log entries**, the overflown log entries are marked as deleted. These entries are in the Audit Log file. The file size will continue to grow.

To delete overflown entries, select **File > Compress**.

- **File > Compress**

This compresses the currently opened audit log file whose log entries are shown in the table. This includes history audit log files. These can be opened using **File > Open**.

When clicking it, the entries that are marked as deleted are removed and the compress action is recorded in the audit log file.

Log history pane

The Log History section is enabled when **Auto cycle log** is selected.

The **Maximum number of log files** option is used to define how many maximum cycled audit log files can be kept in the history folder.

If **Unlimited** is selected, then there is no file number limitation for the cycled audit log files.

When the **Files** option (under **Unlimited**) is selected, the text box is enabled for specifying a limited number. This box cannot be empty and its value must be a positive integer. By default, the maximum file number is 10.

The oldest log file is automatically deleted after the audit log files reach the specified maximum value.

Tracking user activities

The Audit Log can be used to track user activities that happened during runtime. It records the accessing of message data and dumping of message content into the audit log file. With the increase in log information being written into the audit log file, you must to cycle the log file.

The Audit Log can track and cycle in these ways:

- Logging who outputs the extra-long form of message data through the Database Administrator and when it was performed. Optionally, you can log the output message content.
See [Audit Log preferences](#).
- Logging who views and resends SMAT messages and when it was performed. Optionally, you can log the SMAT message data including the message metadata and message content. For resending, only the modified message data is logged.
See [Audit Log preferences](#), [Audit Log Viewer dialog box](#), and [Audit Log detail panel](#).
- Logging who reads the engine log, including which log file name, time and data.
- Sorting log entries by date.
See [Sorting log entries](#).
- Providing automatic cycling ability for the Audit Log.
- Providing a cycle command line for manually cycling the Audit Log.
- Splitting a log entry into multiple entries when the entry size exceeds the maximum limitation.
See [Audit Log preferences](#).

- Encrypting the logged message content of the audit log file to prevent attempts to view sensitive message data.
See [Audit Log preferences](#).

Audit log cycle files

The **File > Open** menu option on the Audit Log Viewer opens the Audit Log file chooser. You can use this to select an audit log file to open.

The audit log file can be an active log file (`audit.log`) or a cycle file. All log cycle files are put into the `LogHistory` folder.

Clicking **Open** loads the selected audit log file into Audit Log Viewer.

If a log cycle file is loaded, then its file name is shown in the dialog box title. **Refresh**, **Preferences**, and **Clear Log** and menus are not available.

If the detail panel is shown, then you can search for specific characters in the detail panel similar to the log table.

- When no entry is selected, the Audit Log Viewer searches from the first cell in the first row.
- When the log entry has detail content, the Audit Log Viewer searches the detail content panel after the message column is searched.

The string that matches the search string is highlighted. The searching order in the detail panel is from top to bottom. In the Metadata table, the searching behavior is the same as the Audit Log table.

The `hciauditcycle` command is provided for manually cycling the audit log file. This command takes no arguments and cycles the current audit log when it is run. This command is included only in host server installs.

The `hciauditcycle` file is installed under the `%HCIROOT%/clgui/bin` folder when installing the host server.

server.ini file

The log configuration of the Audit Server is saved into the `[audit]` section of the `server.ini` file.

```
[audit]
log_size_control_enabled=true
log_auto_cycling=true
log_file_maximum_count=10
log_msg_content=true
log_entry_splitting_enabled=true
log_entry_maximum_size=1000
```

This table shows the log entries and their function:

Log entry	Function
<code>log_size_control_enabled</code>	Indicates whether the log size control for the audit log file is enabled.
<code>log_auto_cycling</code>	Indicates whether automatic log cycling is enabled.
<code>log_file_maximum_count</code>	Specifies the maximum number of log cycle files the Audit Server can contain. "0" is unlimited.
<code>log_msg_content</code>	Indicates if the message content should be dumped into the log file.
<code>log_entry_splitting_enabled</code>	Indicates whether the splitting ability for the log entry is enabled.
<code>log_entry_maximum_size</code>	Specifies the maximum size of the logged message content in one log entry. This takes effect when the <code>log_entry_splitting_enabled</code> field is true. When it is missed or empty, the default maximum value of 1000 kilo-characters is used.

clide.ini file

The show/hide state of the detail panel is saved in `clide.ini` for each user. A key named `audit.viewer.detail.info.visible` is added for this entry.

```
audit.viewer.detail.info.visible=true
```

true indicates showing the detail panel and false indicates hiding the detail panel.

These Audit Log Viewer size keys in `clide.ini` store the sizes of the **Audit Log Viewer** dialog box and the detail panel.

This table shows the available keys:

Key	Description
<code>audit.viewer.height</code>	The Audit Log Viewer dialog box height.
<code>audit.viewer.width</code>	The Audit Log Viewer dialog box width.
<code>audit.viewer.position.x</code>	The X-axis value for the Audit Log Viewer dialog box location.
<code>audit.viewer.position.y</code>	The Y-axis value for the Audit Log Viewer dialog box location.
<code>audit.viewer.divider.location</code>	The width location of the Audit Log Viewer detail panel.

For SMAT detail content, the `audit.viewer.smat.content.encoding` key stores SMAT message content encoding in the Audit Log Viewer. The default SMAT message encoding is the default encoding. A changed value is saved with this key.

Audit log file

The version of audit log file is 3.0. All prior versions of audit log files can be loaded and displayed without problems.

If the current log is an old version, then the log is cycled immediately when the host server starts. New version log entries and old version log entries are not mixed in the same log file.

A detail content flag is written into each log entry that indicates whether the log entry has message content and the message content type. This flag is put after the log message. The message content is appended to the detail content flag.

```
delete flag time source type log message
detail flag message content count length of message content1
length of message content2 ...length of message contentN message content 1
message content 2 ...message content N trailer
```

The format of the SMAT message content is one of these:

- `message format flag metadata_length metadata content`
- `message format flag metadata_length mid content`
- `message format flag mid (only for resend log entry)`

This table shows the content flags:

Flag	Description
<code>detail flag</code>	The content flag, must be one of these values: <ul style="list-style-type: none"> • <code>0x01</code> indicates no detail • <code>0x02</code> indicates SMAT messages • <code>0x03</code> indicates extra-long message outputting
<code>message content count</code>	The total count of the recorded message content.
<code>length of message contentN</code>	The length of the #N message content.
<code>message content N</code>	The content of the #N message that is encrypted.

Flag	Description
<i>message format flag</i>	The content format flag has a fixed length of "4" and must be one of these values: <ul style="list-style-type: none">• 0x10 indicates the detail message includes metadata and content• 0x20 indicates the detail message include content• 0x30 indicates the detail message is MID
<i>metadata_length</i>	The length of unencrypted metadata is optional. If not metadata, then the length is fixed at 8 bytes.
<i>metadata</i>	The SMAT message metadata that is encrypted.
<i>content</i>	The SMAT message raw data that is encrypted.
<i>mid</i>	The SMAT message ID.

Logfile of engine events

The engine maintains a log file that records engine output during the current system session. Click **View Logfile** to open the **Logfile** dialog box.

The text between the brackets (for example, `[cmd:cmd:INFO/0: hcimonitor date/time]`) provides this identifying information about the entry:

- The module and submodule that generated the entry, such as `cmd:cmd`.
- The severity level and engine output level of the event, such as `INFO/0`.
- The thread and, if applicable, the message ID, such as `hcimonitor`.

Viewing the MonitorD error log

To view the MonitorD error log, click **MonitorD Log** to view options:

- Alert Log
- MonitorD Error Log
- MonitorD Log

By default, the standard log is selected, which is the log file opened after clicking **View Logfile**.

Log history

The system log mechanism keeps a copy of multiple log files. The engine and MonitorD log files are accessed through the IDE:

- MonitorD log file
Access the MonitorD log file through the Site Daemons.
- Engine log file
Access the engine log file through the Network Monitor (**Process Controls** dialog box or the context menu for process management).

Automatic log cycling

Configure automatic log cycling through the **Process Configuration** dialog box of the Network Configurator.

Log history features

Log history features include:

- Log history is a configurable feature for engine and MonitorD log files.
- If the log history feature is enabled, then more than one version of the log file is kept in the system for tracking.
- The maximum number of log files is configurable.
- The maximum size of total old log files is configurable.
- The checking on maximum number and maximum size of total log files is performed when the log is cycled.
- A time stamp is given to every record in the log file.

These features apply to these log files:

- Engine logs and engine error logs
- `hcimonitor` logs and error logs

These features do not apply to GDBM and host server logs.

When the **Log History** feature is disabled, log recycling is the same as previous versions.

Configuring log history features

To enable or change the log history features, look for the `LogHistory` lines in the `siteinfo` file. To make changes from the default, you must manually edit these options:

- `LogHistoryEnabled=0`
This number identifies whether the `LogHistory` feature is enabled for this site. This setting applies to the engine and MonitorD logs in this site.
 - `0` indicates the log history feature is disabled. This is the default.
 - `1` indicates the log history feature is enabled.
- `LogHistoryMaxLogFiles=0`
This number identifies the maximum number of backed up log files that are kept if `LogHistoryEnabled=1` (enabled). This setting applies to the engine and MonitorD logs.
 - `0`, or a negative value, indicates the maximum number is unlimited. `0` is the default.
 - `x` indicates the maximum number. This is a positive value.
- `LogHistoryMaxDiskUse=0`
This number identifies the maximum size, in bytes, of the total backed up log files that are kept if `LogHistoryEnabled=1`. This setting applies to the engine and MonitorD logs.
 - `0`, or a negative value, indicates the maximum number is unlimited. `0` is the default.
 - `x` indicates the maximum number. This is a positive value.
- `LogHistoryCompressCommand=NULL`
This string value identifies the command to compress the backed up log files to save the disk space. The value of this option is the binary name to compress the backed up log files. The default value is `NULL` (compression is disabled).

LogHistory directory

When log history is enabled, a `LogHistory` directory in the corresponding `process` or `hcimonitor` directory is created to store all the backed up logs.

`hciengine` creates the directory `$HCISITEDIR/exec/processes/process_name/logHistory`.

`hcimonitor` creates the directory `$HCISITEDIR/exec/hcimonitor/logHistory`.

How multiple log files are labeled

When log history is disabled (`LogHistoryEnabled=0`), a copy is kept of the last engine log file by appending an `.old` extension to the file name.

Only one log file is kept. After the engine is restarted, the `.old` log file is overwritten.

When the log history feature is enabled (`LogHistoryEnabled=1`), multiple log files can be retained.

Using `hciengine` as an example, when log history is enabled, after you start the engine, the engine checks if a `.log.old` file exists.

- If one exists, then the engine renames the `.log.old` log file to `processname timestamp.log` and moves it to the `process_name/logHistory` directory.
- If one does not exist, then the engine checks if a `.log` file exists. If one does not exist, then it goes on without log history.

If one does exist, then it renames the `.log` log file to `processname timestamp.log` and moves it to the `process_name/logHistory` directory.

After being moved, the backed up log file is compressed to save disk space when `LogHistoryCompressCommand` is enabled. Then, a new `.log` file is created.

Log history checking and clean-up

When the engine is started, log history checking is triggered. At that time, the number and total size of backed up log files are checked against `LogHistoryMaxLogfiles` and `LogHistoryMaxDiskUse`. These are defined in the `siteInfo` file.

- If the backed up logs file size is greater than `LogHistoryMaxDiskUse`, then the oldest backed up log file is deleted until the condition fails.
- If the actual number of backed up logs is greater than `LogHistoryMaxLogfiles`, then the oldest backed up log file is deleted until the condition fails.

When engine is recycled, the log history checking follows the same process.

The effective value of `LogHistoryMaxLogfiles` and `LogHistoryMaxDiskUse` is the value in the `siteInfo` file when engine was started. There is no effect if the value is changed after the engine is started.

Log file timestamp

When generating the log file, `hciengine` supplies a time stamp for every record in the log file. The format of time stamp is `month/day/year hours: mins: secs`.

Configuring JMX on Apache Tomcat

This topic describes how to enable JMX on Apache Tomcat in Cloverleaf. The JMX protocol can remotely monitor and manage Apache Tomcat.

Note: This is for Cloud-only installations.

1 Enable JMX on Apache Tomcat in Cloverleaf.

- a Start the Host Server/Security Server and open **Server Administration**.
- b On the **Advanced** tab, configure the **JVM Arguments**. The JVM argument is:

```
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=%my-jmx-port%
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.authenticate=false
```

%my-jmx-port% is the port number that you can modify.

Example: The JVM arguments configured in **Server Administration** would be:

```
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=5551
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.authenticate=false
```

Note: If the Host Server and Security Server are installed on the same machine, ensure *%my-jmx-port%* is different. If this port is not different, then the Server cannot successfully start.

- c Click **Save**.
 - d Restart the Host Server and Security Server
- Verify the Host Server and web service successfully start, or the Security Server successfully starts.
- This JVM arguments must be certified on CIS19.1.0.0.

2 Connect to the Tomcat JMX using Jconsole.

- a Run `jconsole`.
- b Select **Remote Process** and enter the server Host and port number
- c Click **Connect**.

Note: If your Jconsole and Tomcat are not on the same machine, then ensure the firewall is closed.

CPU affinity in Cloverleaf runtime (Windows/Linux)

CPU affinity enables binding a process or multiple processes to a specific CPU core, or range of CPUs, so that processes only run from that specific core. When trying to perform performance testing on a host with many cores, you can run multiple instances of a process, each one on different core. This enables higher CPU utilization.

Cloverleaf supports CPU affinity configuration on the Cloverleaf engine, Monitor Daemon, Host Server, Security Server, and Cloverleaf Windows service.

Configuration

The CPU affinity configuration is located at `$HCIR00T/rootInfo`. This is a key-value pair, where:

- Key name is `cpuaffinity`.
- Value is the list of available processor or core number IDs on the host, separated by a comma.

Example:

```
version=...  
:  
cpuaffinity=0,1,3,5,6,7
```

Runtimes and the logs

After `cpuaffinity` is set in the `rootInfo`, Cloverleaf runtimes follow it to enable CPU affinity. The corresponding information is logged in the runtime log file.

For example, `cpuaffinity` is set in `rootInfo` as:

```
cpuaffinity=0,2,3
```

The `hciengine` log contains:

```
[prod:prod:INFO/0:helloworld:12/11/2020 15:37:00] CPU affinity set is 0,2,3  
[prod:prod:INFO/0:helloworld:12/11/2020 15:37:01] Applying E0 config: 'enable_all'
```

The runtime CPU affinity status is verified by the host.

- On Windows, in addition to the Task Manager PowerShell, the `Get-process` command also has the path.
- On Linux, use the `taskset` command.

Example:

```
PS C:\WINDOWS\system32> Get-Process hciengine | Select-Object ProcessorAffinity
ProcessorAffinity
-----
13
```

When `cpuaffinity` is set incorrectly in `rootInfo`, error information is logged in the runtime log, and runtime does not start.

```
[prod:prod:INFO/0:_hcimonitor:_01/19/2021 19:49:24] CPU affinity set is 0,1,2,x
[prod:prod:ERR /0:_hcimonitor:_01/19/2021 19:49:28] CPU affinity setting is incorrect: 87
```

The Cloverleaf Windows service CPU affinity does not contain the log file. The status is information or an error that is logged in System Event. This is viewable in the **Event Viewer** tool.

The difference is when the `cpuaffinity` setting is incorrect. When this happens, the Cloverleaf Windows service reports an error in **Event Viewer**, and proceeds further without stop.

Reference guide

Engine components

The basic engine components are sites, processes, threads, and queues. Sites contain processes, processes contain threads, and threads wait in queues.

Sites

The processes inside the system run on a single, physical machine. A machine can contain multiple sites, or centers of communication, within a system installation.

Site communication happens with network-based transports and physical connections.

Processes

One site may contain multiple processes working together on message movements. A basic process consists of these threads:

- Inbound protocol thread
- Outbound protocol thread
- Translation thread
- Command thread

The engine contains one or more processes. Each process's data flow is transparent within the engine. Data flow issues are transparent to the user whether they are moving within a single process or between multiple processes.

Note: On an Symmetric MultiProcessing (SMP) server, the only way to run multiple threads in parallel is to put them into separate processes. Threads in the same process cannot run in parallel.

Interthread communication

The engine has an Interthread Communications Library (ICL) that lets threads communicate with each other, even across multiple processes.

Intraprocess messaging takes advantage of a shared address space between threads in the same process. It is also fast, because it does not require data copying or calls from the system to deliver a message.

Threads

Processes contain threads, which are single strands of binary within a process. Threads are the basic units used to configure and monitor the system.

Thread types are:

- Protocol threads are principally responsible for sending and receiving protocol messages on a connection. These communicate with remote host connections and handle protocol-level issues. Because the protocol thread must perform reliably, the message flow and queues inside the thread are tightly coupled with the database recovery states. They also match the protocol, or transfer rules, of each external system:
 - Inbound protocol: These contact the remote host, read the host's protocol, and pull the original message into the engine.
 - Outbound protocol: These send the completed message out to a remote connection, using the connection's protocol.

- Translation threads translate messages between different formats and perform routing between protocol threads.

All inbound messages transitioned from inbound to outbound states go through the translation thread. The translation thread deals only with engine messages, so there is no protocol driver associated with it.

The translation thread uses route details to perform the transition from an inbound to outbound state.

- Command threads schedule flow inside the engine. There is one command thread per process. They also provide options to control the other two thread classes.

The command thread has these main functions:

- Scheduling the running of the protocol and translation threads. This is independent of the OS-dependent thread implementation below it.
- Responding to external commands.

With this interface, each process command thread can accept commands from both local and remote processes. It can also act directly on some commands, and pass the remaining commands along to other threads in the process.

The thread classes provide a pipeline architecture with each thread focused on its own area of processing within the engine.

For example, because protocol threads deal with protocol-level issues, they do not handle record parsing, message routing, or translation systems.

Multiple processes are handled transparently, so that multiple translations may be performed simultaneously.

Message flow between threads

Messages arrive by the inbound side of the protocol thread. From there, they are routed and translated through the translation thread before departing through the outbound side of the protocol thread.

Message flow is described more thoroughly in these topics:

- [Message flow](#)

- [Interprocess message movement](#)

Queues

A queue is an engine internal data structure that stores messages in a given sequence. As messages flow through the engine, they flow through the queues.

A FIFO (First In First Out) sequence is used within a given message priority. Messages that are placed into a queue with the same priority are removed from the queue in the same order: oldest first.

- Inbound pre-TPS queue:

Pre-TPS queues are used only if there are Tcl procedures to run. When the protocol driver has successfully read a message from a connection, it places the message into the Inbound pre-TPS queue. When an inbound pre-TPS runs, it pulls the message from this queue.

If database logging is turned on for this inbound connection, then the message is placed into the recovery database at this point. If the message is not put into the database at this point, then it is not recoverable unless custom Tcl code defines it as recoverable. After the message is placed in the recovery database, it moves forward through each subsequent processing phase within the database.

- Inbound post-TPS queue:

The engine places any inbound messages produced by a TPS into the inbound post-TPS queue. From there, they are sent by ICL to the translation thread within the same process as the inbound protocol thread.

- Pre-translation queue:

The engine places messages that arrive from any protocol thread into the pre-translation queue. When the translation thread runs, it takes the first message from the queue, oldest message first, and begins the translation and routing process.

- Route detail queue:

The route detail queue ensures that all messages from a given route detail are collected together until the current route detail finishes running. This is only used with SEND or CONTINUE. After the route detail completes running, the engine moves messages in this queue to the partial queue.

Primarily, the route detail queue holds messages that result from a translation that produces multiple messages. The goal is to give the user control, by translation UPoCs, over what should happen during a translation failure. The `xperror` command defines different error severities that control the flow of messages from this queue during abnormal translation termination, such as a power outage.

- Partial queue:

The partial queue ensures that the engine performs atomic translations. The engine does this with a two-phase commit of messages. After all route details are run for a message, and any resulting messages are collected in the partial queue. The inbound source message is removed from the recovery database. The engine then begins moving messages from the partial queue to the post-translation queues.

If the engine is interrupted, then it can determine whether it was in message movement to the post-translation queues or in the process of translation. It does this by searching the recovery database for the inbound message.

- If it finds the inbound message, then the engine starts the routing and translation over again.

- If it does not find the message, then it continues moving messages from the partial queue to the post-translation queue.

This queue continues the engine semantic of all messages or no messages resulting from routing and translation, even across unexpected events such as power loss.

- Post-translation queue:

When all of the source message translations are complete, the engine removes the source message from the database. The engine moves the messages waiting in the partial queue to the correct post-translation queue. There is one post-translation queue per destination thread.

After the engine places messages in the post-translation queue, the translation thread attempts to deliver them to the associated protocol thread. Usually, delivery is successful unless the destination protocol thread is stopped, or the process that should contain the thread is not running. If delivery is not successful, then the messages remain indefinitely in the post-translation queue in sequence.

- Outbound pre-TPS queue:

The outbound pre-TPS queue is skipped if there are no outbound TPS procedures to run. Messages sent from the translation thread are placed here for the outbound protocol thread by ICL. When the outbound TPS is ready to process a message, it pulls it from this queue.

- Outbound post-TPS queue:

When the outbound TPS produces one or more messages, they are placed in this queue. The messages remain in this queue until the engine can deliver them to the destination connection, as defined from the protocol thread.

Tcl interpreter

Tcl procedure streams move messages through the translation and routing process. They are code fragments, or pieces of code that fit into another binary, written in Tcl (Tool Command Language).

Tcl is a non-proprietary, full-featured language that controls and extends applications. Tcl uses generic programming facilities, such as variables, loops, and procedures, that are useful for a variety of applications. Because Tcl is interpreted, code can be written on one platform and then moved to another without modification.

Custom Tcl extensions are provided for customization of messages within the engine through UPoCs (User Points of Control). Use these UPoCs to modify message content and message routing.

There is no limitation that Tcl variables cannot contain embedded nulls. The `msgset`, `msgget`, `datget`, and `datset` Tcl extensions can handle binary data. They cannot stop processing when a null character is seen in the data stream.

Upgrading from versions earlier than Tcl 8.0 could cause the Tcl and TclX commands to change some of their behaviors:

- `-lrange`
- `-file dirname`
- `-keylget`
- `-translit`

Scripts that are created in versions of Tcl earlier than Tcl 8.0 might not run properly in Tcl 8.4 and should be reworked.

Tcl interpreter

For translate or generate route details, each procedure in the sequence acts independently. Each procedure is invoked with the same keyed list interface and expected keyed list output. In all TPS, the same Tcl interpreter is used to invoke all procedures. They can use the same Tcl global variables or external database state.

If any procedure in the list returns more than a single message, then all procedures after it are called repeatedly, once for each message. Procedures returning multiple messages do not track more than a single message. They can use, though, Tcl global variables or database states to keep track of message source mid values.

Any messages that are generated by previous procedures in a raw detail or by previous route details are still delivered according to their configuration.

The pre- and post-processor calling sequence is different than all other Tcl interfaces in the system: more context is created before the processor is run, so that it can use a Tcl code fragment for the processor. Tcl code fragments can also contain global variables that are available in both inbound and outbound threads of the same process.

In-progress translations that are found during translation thread start-up are restarted from the beginning. The same Tcl interpreter is used for all Tcl callouts during a single message. It is also used for all translations that are performed by a single translation thread.

Note: Exercise caution when passing state information between route details.

When the system supports more than one translation thread, it is almost impossible to pass Tcl state information between the translations of different messages. There is one Tcl interpreter per translation thread, so that there are different Tcl states for translation threads.

In such an environment there is no guarantee that a single Tcl interpreter is used to translate all source messages from a given source connection. This is because parallel translations translate messages concurrently from the same connection.

`hcigdbmconvert` promotes a GDBM file from a version earlier than 3.8.1P rev1. This utility converts null-terminated databases to non-terminated, and resides in `integrator/bin`.

Upgrading from Tcl 7.6

The core of the Tcl interpreter has been replaced with an on-the-fly compiler than translated Tcl scripts to bytecodes, resulting in faster running of code. In Tcl 8.X, strings are replaced with `Tcl_Obj` structures ("objects"). These can hold both a string value and an internal form such as a binary integer or compiled bytecodes. The new objects make it possible to store information in efficient internal forms. This avoids the constant translations to and from strings that happened with the old interpreter. Sometimes, particularly errors, this can cause scripts to behave differently than in previous versions of Tcl.

Tcl 8.X causes a few changes to Tcl/Tk scripts or C extensions. Most of these changes let Tcl programs run faster.

These changes are:

- Compilation errors return errors immediately, before running a script.

Compilation errors are returned for scripts with malformed expression words. For example, words with missing close braces or quotes. In addition, several commands are treated specially by the Tcl bytecode compiler and are compiled into an in-line sequence of instructions. This is for better running speed.

In Tcl 8.X, these commands are:

- `break`
- `catch`
- `continue`
- `expr`
- `for`
- `foreach`
- `if`
- `incr`
- `set`
- `while`

If a script includes one of these commands with the incorrect number of argument words, then a compilation error is returned immediately. At this point, the script is not run. For example, `set a xxx yyy`. By returning a compilation error immediately, you can find typos and other syntax errors in your scripts as soon as possible.

- `catch` no longer catches compilation errors.

Because compilation errors in scripts are returned immediately, `catch` no longer prevents compilation errors from stopping command interpretation.

For example, `catch {set} immediately reports the error wrong # args: should be "set varName ?newValue?".`

To have `catch` handle compile time errors, give a script to it that is only determined at runtime. For example `catch [concat {set}]`.

- Scripts are completely compiled.

Tcl 7.6 would ignore any characters in a script after the last command run. This would let you, for example, put comments after the `exit` command that terminated a script without having to start each comment line with a `#`.

For example, previously you would write:

```
initialize ;# the commands
compute ;# of the
finalize ;# program
exit ;# stop execution
This is the first line in the script's change log.
A second line in the change log.
```

Tcl 8.X attempts to compile those lines. You should put a `#` at the start of each comment line.

- Lists are aggressively parsed.

Lists are a real, and significantly faster, data type and not strings that are rescanned each time. They are used the same way, but strings are converted to a faster internal representation behind the scenes of the first list operation. Lists are now completely parsed.

Tcl 8.X reports an error when you have a malformed list, and the erroneous part was after the point an element was inserted or extracted.

For example, in Tcl 8.X:

```
lindex {a b "c" d e} 1
```

This returns the error `list element in quotes followed by "d" instead of space`.

In Tcl 7.6 it returns *b*.

- List operations do not preserve the exact white space between elements.

List operations in Tcl 7.6 always retained the white space between list elements. In Tcl 8.X, list operations return lists whose elements are separated by a single space.

In this example, the first command sets the variable *x* to a string containing two tab characters:

```
set x {one two three} lrange $x 0 1
```

In Tcl 8.X, the `lrange` command returns `"one two."` Programs that preserve white space should use string operations or use a combination of the `split` and `join` commands.

For example:

```
set x {one two three}
join [lrange [split $x{ }]0 1] { }
```

This preserves the tab between the list elements. There is a tab character inside each pair of braces.

- Fewer floating-to-string conversions, and the associated rounding, can change program behavior.

Both Tcl 8.X and 7.6 use full IEEE floating-point precision, about 17 decimal digits, when computing expressions. They both round floating-point values when converting them to strings. By default, Tcl 8.X retains 12 digits and Tcl 7.6 keeps 6 digits. The new object system in Tcl 8.X causes fewer floating-to-string conversions, and the associated rounding, to happen than in Tcl 7.6. These changes mean that floating-point computation is more accurate and faster in Tcl 8.X, but there are sometimes behavioral changes.

For example:

```
for{set x 0.0}{$x != 4.0}{set x [expr $x+0.1]}{puts $x}
```

This terminates in Tcl 7.6 but loops forever in Tcl 8.X. This is because the fraction `0.1` cannot be exactly represented as an IEEE floating-point value. Repeatedly adding it to `0.0` never produces a floating-point value that is exactly `4.0`. This loop terminated in Tcl 7.6 only because of the rounding that was performed at every assignment to the variable *x*.

The solution in Tcl 8.X is to use approximate comparisons for floating-point. When looping until the value of a floating-point expression reaches a target value, test whether the expression is "close enough" to the target.

For example:

```
for{set x 0.0}{abs($x-4.0)>0.001}{set x[expr $x+0.1]}{puts $x}
```

This stops when the variable *x* reaches `4.0`.

- The original strings in expressions are retained.

For example:

```
expr{"0y"<"0x3"}
```

This yields 0, not 1, in Tcl 8.X because the original string for `0.3` is not lost. Tcl always tries to convert expression operands to numbers when possible. In Tcl 7.6, the string `0x3` was lost when being converted to a number (`3`) by the expression implementation and then back into a string. At this point a string comparison had to be performed. This meant Tcl 7.6 did a string comparison between `0y` and `3`. Because Tcl values are now stored in Tcl objects that hold an internal form and a string, the original string is not lost. Tcl 8.X can do the string comparison between `0y` and `0x3`.

- `info cmdcount` is no longer accurate.

Compiled commands are compiled away and their running is no longer counted. In Tcl 8.X, this includes the commands:

- `break`
- `catch`
- `continue`
- `expr`
- `for`
- `foreach`
- `if`
- `incr`
- `set`
- `while`

- `append` no longer triggers read traces.

The `append` command no longer triggers read traces when retrieving the old values of variables before doing the append operation. It only triggers a write trace after each append.

The `lappend` command still triggers read traces in this situation. `lappend` has changed to behave as `append`.

- `lappend` triggers only one write trace after appending all arguments to the list.
- Error tracebacks are shorter.

There are fewer recursive calls to `Tcl_Eval` because some commands such as `while` and `if` have been compiled into a sequence of bytecoded instructions. Tracebacks often have fewer traceback levels than in the past.

- The length of local variable names does not matter.

For most commands, the compiler looks up local variables in a procedure at compile time. Then, it emits instructions that, at run time, take the same amount of time regardless of the length of the variable names.

Route detail types

Route detail types consist of:

- **Raw:** This tells the engine to pass the message data through without translating it.

- **Translate:** This translates the message according to the actions in the specified translation file.
- **Generate:** This processes the message through a sequence of specified Tcl procedures to generate new route details.

Raw route detail type

The raw route detail type tells the engine to pass the message data through without translating it. As with all route types, the message can be processed by one or more Tcl procedures.

The raw route detail takes a single source message and turns it into resulting messages going to one or more destination protocol threads. The resulting messages can contain user data and metadata as the source, or they can contain different data.

A raw route detail may be made up of a list of Tcl procedures, which are run in sequence. If the procedure does not return a message, then the translation of the source message ends. The source message that is given to the procedure is not moved to the protocol or post-translation queue, and the message dies.

Upon entry to each procedure, the source message's translation destination metadata field contains the list of the message's destination connections.

Before the first procedure is called, the source message's metadata is set to the default list of destinations that is configured for the route detail. Each procedure can modify the list of destination connections, but if no modifications are made, the default list stays in place. Modifications to the list can be made by direct manipulation or by calling a programmatic interface.

Each message returned by a procedure may or may not contain the same data and destination connection information. A procedure can create multiple messages with different data, with each going to one or more different destination connections.

When the engine has called all procedures that are listed in the raw route detail, each message returned is placed into the Protocol Message Queue. After the entire translation is complete, each message is moved to the Post-Translation Queue. Then, the message is sent to the destination connection that is listed in the message's translation destination data. This is in the **msgXlateDests** field.

If a raw route detail must generate a message unrelated to the flow of the source message, then it can create a binary object that:

- Updates the new message's translation destination data in the **msgXlateDests** field to point to the destination for the new message.
- Returns the original, possibly unmodified, source message.
- Returns the new unrelated message.

Unrelated messages are not given to any remaining procedure associated with the route detail.

For example, a message that is given to the remaining procedures is tagged with a key of `CONTINUE`. The message sent to a connection independent from other messages is tagged with a key of `SEND`.

These messages are held in separate keyed lists, a Tcl data structure. A message's disposition change sets it on a new path, separate from its peers.

Any message that is listed in a keyed list is immediately placed into the Partial Queue without further processing.

After the translation is complete, all messages in the Partial Queue are moved to the correct Post-Translation Queue in a FIFO message priority order. `CONTINUE` and `SEND` messages follow the same path as other messages, but the `CONTINUE` messages go on to further post-processing.

If the translation does not finish for any reason, then messages that are placed in the Partial Queue are not sent to any connection.

If the source message still exists in the Pre-Translation Queue, then messages in the Partial Queue are removed, and the translation restarts.

Common metadata field names are:

- Message ID: `msgSrcMid`
- Message translation destination: `msgXlateDests`

Translate detail type

The translate detail type translates the message according to the actions in the specified translation file. This is also referred to as `xlate`.

The translate route detail has these main parts:

- Pre-processors. These are code fragments that act on a message before translation. This is not required.
- Translation. This performs one translation containing translation actions. This is required.
- Post-processors. These are code fragments that act on a message after translation. This is not required.

Each translation is made up of translation actions. Sometimes there are zero translation actions; often there are many more. Each action in the translation can have a pre- or post-processor. The pre- and post-processor calling sequence is different than all other Tcl interfaces in the system: more context is created before the processor is run, so that it can use a Tcl code fragment for the processor. Pre- and post-processor Tcl callouts must be written using the input and output API of the system. The pre- or post-processor affects the translation process by modifying local Tcl variables in the code fragment.

These local variables are supported:

- Translation In Values or `xlateInVals`
- Translate In Types or `xlateInTypes`
- Translation Out Values or `xlateOutVals`
- Translate Out Types or `xlateOutTypes`
- Translate In List or `xlateInList`
- Translate Out List or `xlateOutList`
- Translate Send Message or `xlatesend`

The pre- and post-processors have the same interface and power described in the raw route detail procedures. Each procedure can use the `CONTINUE` and `SEND` keyed lists to process messages.

If pre-processor run results in more than one message, then the translation and the post-processor are called in sequence for each resulting message. Both the `xlate` and the post-processors do not track multiple messages unless they track the message's source ID (`msgSrcMid`) value of messages. Between the pre- and post-processors, a single message is translated using a `.xlt` file extension.

If a pre- or post-processor needs to modify the destination connection information, then it modifies a Tcl variable named `XlateDests`. These are translation destinations. As with the raw route detail, these modifications are made by direct manipulation or by calling a programmatic interface.

If a processor needs to generate a message unrelated to the source message's flow, then it:

- Creates a binary object.
- Updates the new object's translation destination that is specified in the `XlateDests` field to point to the appropriate destination.
- Modify a Tcl variable named `XlateSend`.

The contents of this variable is a keyed list with a key of `SEND`.

After the action's pre-processor or post-processor code fragments are complete, the xlate thread checks the values of all the translation, or Xlate, variables. Messages pointed to by the `XlateSend` sending translation variable are moved to the Protocol Queue. The movement and delivery semantics of these messages are the same as messages returned by procedures in a raw detail using the `SEND` keyed list.

An invocation operation can also be specified for an action. This operation is a Tcl code fragment and has the same calling semantics as the action pre- and post-processors. This functionality gives complete control over the translation action.

Generate detail type

The generate detail type processes the message through a sequence of specified Tcl procedures to generate new route details.

This route detail only controls a message route, using a single user-written Tcl procedure. It does not usually modify the source message or directly change its metadata, although it has that option.

- The input for a generate procedure is a keyed list of remaining route details and the source message.
- The input to a generate procedure is a keyed list of remaining route details. Completed route details are not included in the list.

The generate detail can:

- Modify the input detail list. For example, it can insert new details into the list or remove existing details.
- Add additional generate details to the route detail list that are called when processed.

Return value

The return value contains an updated route detail list and any message that should be processed using the route detail list:

- If multiple messages are returned, then the messages are processed in sequence by the returned list of route details.

- If no message is returned, then the translation of the source message ends. The source message that is given to the procedure is not moved to the protocol thread or post-translation queue. This stops the translation. Messages that are generated by previous route details can still be delivered.
- In some instances, one or more messages and a route detail list containing one or more details are returned. In these instances, each message is given in sequence to the route details.
If the routine returns no messages and a non-empty route detail list, then it is considered an error.
- If a route detail list is not returned but one or more messages are, then the translation of the source message ends. The message that is returned by the procedure are moved to the partial queue.

Note: The route detail list that is returned applies only to the current message and any subsequent messages that are derived from the current message. When a new message is removed from the pre-translation queue, the route detail list that is defined in the NetConfig file is used.

State information

Previous translation activity that results in Tcl code being run can store state information for use during a generate route detail.

For example:

- Raw route detail procedures
- Pre- or post-processor procedures for a translate route detail
- Generate route detail procedures
- Pre- or post-processor code fragments for a translate route detail action

State information can be in the form of Tcl global variables or a database.

Note: Unless state information needs to be saved across source messages. There is no reason to put it in a database.

Message flow

Inbound messages are received by the protocol thread (inbound), transported to the translation thread, and exported through the protocol thread (outbound).

Inbound protocol message flow

The normal flow of inbound messages inside the protocol thread is:

- 1 Protocol messages arrive from the protocol driver.
- 2 The engine places the protocol messages into the inbound pre-TPS queue.
- 3 The messages undergo TPS processing in the appropriate inbound TPS.
- 4 The engine places the messages into the inbound post-TPS queue.

- 5 The engine sends the messages to the translation thread within the same process.

Translation thread message flow

The normal flow of messages inside the translation thread is:

- 1 Untranslated engine messages arrive by ICL from the inbound protocol thread.
- 2 The engine places the messages into the pre-translation queue.
- 3 The messages pass through routing and translation.
- 4 The engine places any resulting engine messages into the route detail queue.
- 5 After a route detail is complete, any messages in the route detail queue are moved to the partial queue.
- 6 The messages that are created by all route details in the partial queue are moved to the post-translation queue. This is based on the destination and sent to the protocol thread. There is one post-translation queue per outbound protocol thread.

The translation thread receives and sends all messages by ICL. There is no protocol driver associated with the translation thread.

Between the pre-translation queue and partial queue, the message is in memory only.

Outbound protocol message flow

The internal flow of outbound messages within the protocol thread involves many factors, including:

- Time-out and retry management.
- Start-up initialization of the TPS.
- Message forwarding, if required, after they pass through TPS processing.

The normal outbound flow of a message through the protocol thread is:

- 1 A message arrives by ICL from the translation thread.
- 2 The message moves into the outbound pre-TPS queue.
- 3 The message undergoes TPS processing in the appropriate outbound TPS modules.
- 4 Any resulting protocol messages move into the appropriate outbound post-TPS queue.
- 5 The message undergoes any prewrite UPoCs specified, and is then placed in the forward queue.
- 6 The engine attempts to send the first outbound protocol message.

If the attempted send succeeds, then the engine removes the protocol message from the queue.

If the attempted send fails, then:

- Time-out determines if and when the engine resends the message.
- Retry values determine how often the engine resends the message.

If the number of attempted sends exceeds the retry value, then the engine does one of these:

- Place the protocol message in the error database as undeliverable.
- Forward the message if there is a forward host that is defined for the connection.

Reply handling

In some instances, a protocol thread is configured to await replies. The next protocol message that is delivered from the protocol driver should be an inbound reply protocol message.

The engine looks for:

- Metadata in the outbound message sent into the outbound TPS module.
- Configuration information from the `NetConfig` file.

Only one message at a time is moved to the forwarded thread's outbound queue. This is even if the forwarding thread's outbound queue contains multiple messages.

Protocol message forwarding

Messages that are forwarded to a new thread are usually written using the protocol information of the destination thread. The exception is if the forwarded thread is also forwarding messages to another thread, forming a chain of forwarding threads.

These message forwarding commands are called from alerts, the command line (`hcicmd`), or Network Monitor:

- **Forward Start** (`hcicmd -fwd_start`)
This starts forwarding data messages to the thread that is specified by the `FORWARDTHREAD` Network Configurator entry.
- **Forward Stop** (`hcicmd -fwd_stop`)
This stops forwarding data messages.
- **Forward Reply Start** (`hcicmd -fwdr_start`)
This starts forwarding reply messages to the thread that is specified by the `FORWARDTHREAD` Network Configurator entry.
- **Forward Reply Stop** (`hcicmd -fwdr_stop`)
This stops forwarding reply messages.

When forwarding is enabled, the message is moved from the original thread's outbound queue. Then, it is placed in the outbound queue of the forwarded thread.

Before this happens:

- 1 The message's `MSG_FLAG_FORWARDED` flag is set.
- 2 The message is run through the forwarded thread's outbound TPS destination thread, if there is one. If special TPS procedures are to be applied, then use a thread that does not receive non-forwarded messages.

Chaining

A single thread can be both a forwarded thread and a forwarding thread, which supports the creation of an infinite chain of message forwarding threads.

Messages in a forwarding chain move through each thread's outbound data, or reply queue, and any outbound TPS procs that exist.

- `hcicmd fwd[r]_start` operates on only one thread, so it must restart for each link in the forwarding chain.
- `hcicmd fwd[r]_stop` preserves the message order when it shuts down a chain.
- `hcicmd fwd[r]_stop` shuts down current thread forwarding and any later links in the forwarding chain.

Note: Configure a thread to accept only forwarded or non-forwarded messages. Configuring a single thread to accept both forwarded and non-forwarded messages could delay the `hcicmd fwd[r]_stop` command.

For example, A forwards to B, B forwards to C, C forwards to D.

- A `fwd_stop` that is issued to A would shut down A, B, and C forwarding.
- A `fwd_stop` that is issued to B would shut down B and C forwarding.
- A `fwd_stop` that is issued to C would only stop C from forwarding to D.

Disk-based messages

The recovery database stores disk-based messages. Make disk-based messages available at the process or thread level.

The overall memory limit (`MSGSPACELIM`) is configured by selecting **Disk-based Queuing** in the Network Configurator's **Process Configuration** dialog box.

When this is selected, specify the threshold in **Threshold in Megabytes**. This amount must be greater than zero. It indicates the amount of memory that the process allocates for message data before storing messages to the recovery database.

When this is not selected, the threshold field is "0" and no memory is allocated.

Recovery database message flags

This table lists the recovery database message flags:

Message flag	Description
<code>MSG_FLAG_KEEPPONDISK</code>	<p>This is set when a message is created, if the inbound thread is configured to keep messages on disk.</p> <p>This is used to keep message data on disk, in the recovery database, instead of in memory (RAM). It operates the same way as using the recovery database for recovery purposes.</p> <p>After this is set, it remains in effect until the message leaves the engine, or when message delivery is successful.</p>

Message flag	Description
MSG_FLAG_USERDB	<p>This is used to store messages for recovery purposes. After a message is marked for recovery, it is backed by the recovery database throughout its life in the engine.</p> <p>The recovery database backs up the thread and process queues. If an engine process or thread stop running, then it retains the message data from the recovery database when the thread or process is restarted.</p>
MSG_FLAG_ISONDISK	<p>This is used to store messages on disk only, not in memory. If any operator needs to access the message data, and this flag is set, then the system retrieves the message from disk. It is then placed into memory.</p>

Disk-based queuing and VM/DISKQ_PERCENT environment variable

These cases can trigger disk-based queuing:

- **Disk-based Queuing** for the process is selected and the total RAM that is required for the queues exceeds the configured threshold. The message count in queue is more than or equal to the default minimum message count.
- The used virtual memory percent exceeds the threshold. The message count in queue is more than or equal to the default minimum message count.

You can tune the trigger condition by setting the environment variables `DISKQ_PERCENT` and `DISKQ_MINMSGs` before starting a process.

- `DISKQ_PERCENT`: The default `DiskQue` Virtual Memory percent is 75. (75%).
- `DISKQ_MINMSGs`: The default `DiskQue` Minimum # of Messages/que is 50.

Interprocess message movement

There are two processes in this configuration.

- Process 1 contains four threads: two protocol threads, one translation thread, and one command thread.
- Process 2 contains three threads: one protocol thread, one translation thread, and one command thread.

In this interprocess message transaction:

- The message arrives from a connection.

- The message is routed and translated.
- The translated message is delivered to the destination connection.
- The destination connection sends a reply message.
- The reply message is routed and translated.
- The reply message is delivered to the source connection.

A protocol thread is responsible for communicating with one remote host connection. That connection forms part of the protocol thread name. For example, the proto "A" thread is the protocol thread responsible for communication with Connection "A." All protocol messages sent to or received from a connection are handled by the protocol thread that owns the connection.

Engine commands

`hci` commands directly control the engine or the GUIs. These commands access the engine without GUIs and can:

- Start the GUIs outside the IDE.
- Modify the engine's actions.
- Show and modify file contents.
- Customize user notifications, such as alerts and views.

A thorough understanding of the commands and their functions is required. Throughout the section, command input is noted with all options, although not all listed options are necessary in each command.

hci GUI commands

The `hci` GUI commands include:

- `hciaccess`
- `hciauditlog`
- `hciguitcptest`
- `hciserveradmin`
- `hciguisiteinit`
- `hcisitedocgui`
- `hciupgradeutility`

hciaccess

This starts the IDE.

The IDE is also accessible through the Windows Start menu.

hciauditlog

This starts the Audit Log Viewer.

hciguimsg

This shows a user-defined message on the X display.

```
hciguimsg [-input file|-message message] [-title window title]  
[-error|-alert|-fg color |-bg color] [-font fontname]  
[-label window label] [-width width in characters] [-height height in characters]  
[-bell repeatInterval]
```

- `-input` specifies input of the listed `file`.
- `-message` specifies message name.
- `-title` specifies the window title.
- `-error` specifies the message error.
- `-alert` specifies what alert is triggered.
- `-fg color` specifies font color of the message.
- `-bg color` specifies the background color of the window.
- `-font` specifies the specific `fontname`.
- `-label window label` specifies the text that is written on the label above the message.
- `-width width in characters` specifies the width of the message window.
- `-height height in characters` specifies the height of the message window.
- `-bell repeatInterval` specifies how often, in seconds, that the computer's internal system bell rings.

hciguitcptest

This starts the TCP/IP Test tool.

hciserveradmin

This starts the **Server Administration** tool.

hciguiseinit

This starts the Site Init GUI.

```
hciguiseinit [-s][-i] newsite
```

- `-s` starts the daemon processes after site creation.
- `-i` does not enforce site name restrictions.

Note: Generally, the `-i` option is not used by users. Contact support for assistance.

- `newsite` is the name of the new site to create. This site name must not currently exist.

The Site Init GUI can also be started from the Start menu, at **Start > Programs > Infor Cloverleaf Integration Suite > Tools > Site Init**.

hcsitedocgui

This starts the **Site Document** dialog box.

hciupgradeutility

This starts the Security Upgrade utility.

hci non-GUI commands

hci non-GUI commands include:

- `hciaclimport`
- `hcialertchedk`
- `hcialerts`
- `hcialertnotify`
- `hciauditcycle`
- `hciauditlogexport`
- `hcibox`
- `hcicmd`
- and others

hciaclimport

This command creates/updates users and roles, and creates/updates/deletes predefined ACLs. Log messages are appended to the log file stored at `$HCIR00T/security/logs`.

This does not clear the database, but only adds/overwrites to the database. Targeted deletion of rights is permissible.

Users and roles can only be deleted in ACL Role Manager. This tool can only be used on security server.

Command usage is one of these forms:

```
hciaclimport [-f filePath -p password [-b] databasePath]
[-r]] [-t]] [-a tool]] [-h|-help]
```

```
hciaclimport databasePath [-r]
```

- `-f filePath` is the path of the predefined `xml` file.
- `-p password` is the password of CA certificate.
- `-b databasepath` is the path of the database backup.
- `-r` connects to security server. When this command is used during installation and security server is not running, this option should not be used. If security server has already been successfully installed and is running, then you can use `-r` to define the pre-ACLs.
- `-t` lists all supported tools.
- `-a tool` lists all supported actions of the tool.
- `-h` or `-help` prints this help message.

hcialertcheck

This command validates the alert configuration file, *alertFile*. If *alertFile* is not found in `$HCISITEDIR/Alerts`, then the script throws an error.

```
hcialertcheck [-f alertFile]
```

`-f alertFile` is the name of the alert file to verify. If this is not given, then the `default.alert` file is verified.

This is also run by the daemon during loading/reloading of configuration files. Errors prevent the alert from loading. Warnings permit the alert to load, but the alert as defined might not run as expected.

hcialertls

This command lists alert files in a site:

```
hcialertls [-lp]
hcialertls [-l][-p]
```

- `-l` shows the long listing.
- `-p` shows file prologues.

hcialertnotify

The Global Monitor Alert Notify feature requires command-line support to trigger GM to open the alert notification.

```
hcialertnotify [-M message] [-f file_name] [-S source_list]
```

- `-M message` is the alert message.
- `-f file_name` is the text file to send as the notify message. *file_name* can be multiple lines, for multiple line alert messages.
- `-S source_list` is the source list separated by a space. The default is command-line.
For example: `-S "source1 source2 source3"`.

The Global Monitor alert notification window is triggered by one of these methods:

- An alert triggers with notify selected as an alert action.
- A command-line interface is run on the Cloverleaf server that triggers the notification.

The server interface is a command-line interface. This interface includes, but is not limited to, thread name, process name, site, and text.

A list of alerts is kept in `alerts.log`, found at `%HCISITEDIR%\exec\hcimonitor.d`.

hciallowlist

Use this command to automatically allowlist all your custom `jar` files when you upgrade your sites or make changes to your `jar` files. This is also necessary for `class` files.

For example, FHIR Bridge has numerous `jar` files that must be added during installation. This can be run as a script instead of listing all of the files. This is available on all platforms.

Usage:

```
hciallowlist -s site -d directory -r r [-l log_file_path] [-h | -help]
```

Where:

- `-s site` specifies a site. The `jar/class` allowlist works on this site. This is required.
- `-d directory` specifies a directory folder. All `jar/class` files in this folder are added to the java allowlist. This is required.
- `-r r` Attempts to recover the allowlist database to a previous state.
- `-l` specifies the log file path.
- `-h` or `-help` displays this help message.

This command line removes all existing Java allowlists that contain files in the specified folder or sub-folder.

This also scans the specified folder and its sub-folder. All `jar` and `class` files are added to the Java allowlist of a specified site.

Example:

```
hciallowlist -s helloworld C:\infor\test_folder -l C:\infor\addallowlist.log
```

This adds all `jar` and `class` files in `C:\infor\test_folder` and its sub-folder to the java allowlist of the “helloworld” site.

hciauditcycle

This command manually cycles the audit log file. This command takes no arguments and cycles the current audit log when it is run. This command is included only in host server installs.

The `hciauditcycle` file is installed under `%HCIROOT%\clgui\bin` folder when installing the host server.

hciauditlogexport

Note: For security, users must have a cl-aom-sec-advanced key to export the Audit Log.

This feature is accessed only through the command line.

```
hciauditlogexport [-i] [-k filesize in kb] [-f filename] [-ds start date] [-de <end date>]
[-v] -e export filename | [-h | -help]
```

- **-i** is to include the message content.
The default is `$HCIR00T/server/audit/audit.log`. You can also export the cycled audit log by specifying the file name with path.
- **-k *filesize in kb*** limits the export file size.
- **-f *filename*** exports an existing audit log. *filename* is the file name of the audit log with full path.
- **-ds *start date*** exports the records which are from this date. The date format is mm/dd/yyyy.
- **-de *end date*** exports the records which are up to this date. The date format is mm/dd/yyyy.
- **-v** prints out verbose information.
- **-e *export filename*** exports the file name.
- **-h** or **-help** displays this help message.

hcibox

This command automates BOX export, import, and deployment.

```
hcibox {-i filepath | -e filepath | -d sitename | -c
sitename | -s sitename | -n sitename thread
[, thread...] [[r] [m] [p]] | -r [[i | s | r] [a | d]]} BOX [-f] | [-v] |
[-h | -help] | [-x cfgfile=configurationfilename] [-l]
```

The corresponding binary is found under `%HCIR00T%/clgui/bin` and is only for host server installations.

- **-i *filepath*** imports a *BOX* into the current host.
filepath is the full path to the imported *BOX*.
- **-e *filepath*** exports an existing *BOX* as a file package.
filepath is the full path to where the *BOX* is exported.
- **-d *sitename*** deploys a *BOX* to the specified site.
sitename is the name of the site to where the *BOX* is deployed.
- **-c *sitename*** creates a new site with a *BOX*.
sitename is the name of the site to be created.
- **-s *sitename*** creates a new *BOX* packaging the entire site.
- **-n *sitename thread*, [*thread...*] [[*r*] [*m*] [*p*]]** creates a new *BOX*.
 - *sitename* is the name of the destination site.
 - *thread*[, *thread...*] is the name list of the threads to be referred.
 - *r* automatically includes all referenced threads.
 - *m* includes master site/root resources when finding references.

- `p` packages master site/root resources as their original location into *BOX*.
- `-r [[i | s | r][a | d]]` refreshes the *BOX*.
 - `[i | s | r]` indicates that when refreshing a *BOX*, what to do if some *BOX* resources do not exist on the target environment.
 - `i` ignores the resource and continues to refresh others. This is the default.
 - `s` stops the refresh and prompts an exception.
 - `r` removes the resource from *BOX*.
 - `[a | d]` indicates how to deal with new referenced resources found in site/mastersite/root.
 - `a` adds new referenced resources into *BOX*. This is the default.
 - `d` discards the new referenced resources.
- *BOX* is the *BOX* location and name. The location is the relative path to the %HCIR00T%/box folder.
- `-f` forces running, skipping the query.
- `-v` out verbose information.
- `-h` or `-help` displays this help message.
- `-x cfgfile=configurationfilename` chooses the protocol configuration file in the current *BOX*. This is only for deploying a *BOX* or creating a site from a *BOX*.
- `-l` disables route level deployment when creating a *BOX* using the command line.

hcidcmd

When the engine is running, `hcidcmd -p process` brings up an `hcidcmd` prompt. Use the prompt to communicate with the engine processes command thread.

```
hcidcmd -p process [-s site] [-t type]
[-h host] [-c cmd...] [-v] [-w seconds] [-help]
```

Note: Access commands are also available through the Network Monitor GUI.

- `-p process` specifies the engine process name. This is optional if `-p` is `d`.
- `-s site` specifies the site.
- `-t type` specifies the type of process:
 - `e` specifies engine. This is the default.
 - `d` specifies daemon.
- `-h host` specifies the process host machine.
- `-c cmd` runs the command.
- `-v` specifies verbose mode.
- `-w seconds` sets the time-out. The default is 30 seconds. The input value range is 1,65535.
- `-help` shows usage help message.

`-t d` arguments

These are used to send commands to the running monitor daemon.

The `-p` argument is optional when `-t d` is specified.

To run a single command, invoke it using the `-c` option:

```
hcicmd -t d [-p hcimonitor] [-c command]
```

To run in an interactive mode, omit the `-c` option:

```
hcicmd -t d [-p hcimonitor]
```

cyclestatsdb

This manually cycles statsDB.

For example:

```
D:\>hcicmd -t d -c "cyclestatsdb"
```

Response:

```
0: cyclestatsdb (Stats DB cycled)
```

Another response is:

```
1: cyclestatsdb (Stats DB is opened and cycling failed)
```

Accompanying syntax

From the `hcicmd` prompt, the accompanying syntax can be run:

- `conn client_drop conn ID:`
Disconnects a client that is based on the connection ID.
- `conn client_list:`
Shows a list of active client connection IDs. If **Save Client IP and Port to DriverControl** is enabled on the **Multi Server Options** dialog box of the TCP/IP or PDL TCP/IP protocol, then this shows the IP and port of each client.
- `dbrate secs:`
Changes the default sample rate of the error database alerts from three minutes to the value that is given by `secs`. This is the number of seconds that are between samples.
This command does not permit the setting of this interval to less than 60 seconds. If an interval less than 60 seconds is specified, then an error message is returned and the value remains unchanged.
This changes the polling rate of all `errdb` type alerts.
- `.die:`
Stops the engine.
- `disablefilter:`
Disables alerts that match the specified filter. This does not reset an alert, nor does it reset times that are used in alert tests.

The time-out for the alert to fire is a "test time" of the alert. When it is enabled, that time is used in the test. This continues until a new value is set by the thread or the thread is restarted.

In general, do not clear the disable until you are sure that an event changes the time in msi or that the alert may fire. In most cases, this is after the thread has been bounced.

filter can be one of:

- `all` disables all loaded alerts.
- `alert name` disables the named alert.
- `group group_name` disables alerts belonging to group *group_name*.
- `config file_name` disables alerts loaded from the configuration file *file_name*, and where *file_name* is the alert configuration file, for example, `default.alrt`.

- `enable filter:`

This enables alerts that match the specified filter.

filter can be one of:

- `all` enables all loaded alerts.
- `alert name` enables the named alert.
- `group group_name` enables alerts belonging to group *group_name*.
- `config file_name` enables alerts loaded from the configuration file *file_name*, and where *file_name* is the alert configuration file, for example, `default.alrt`.

In some situations, an alert is a member of more than one group and more than one group has been disabled. In this case, only the group named in the command list is enabled. For example, an alert is a member of lab-a and clinic-b, and both were disabled by the `disable group` command. `enable group lab-a` still leaves the alert disabled as a member of clinic-b. The named and all versions of the command override this type of action.

- `conn eo_alias alias:`

Supplies output alias.

- `conn eo_en eoc_pat:`

Enables trace output.

- `conn eo_dis eoc_pat:`

Disables trace output.

- `conn cycle:`

Cycles the Daemon output file.

When using `cycle`, *process* is not an engine process, but the MonitorD.

- `conn fwd_start:`

Starts message forwarding.

- `conn fwd_stop:`

Stops message forwarding.

- `conn fwdr_start`

Starts reply message forwarding to the thread that is specified by the `Forward Thread` entry in the Network Configurator.

- `conn fwdr_stop:`

Stops reply message forwarding.

- `log message`:
Writes *message* to the `hcimonitor.log` file and to the `alerts.log`. This command assumes that there are no escaped characters in *message*. Example:

```
timestamp : User Log Command: user log message
```

- `output_cycle`:
Cycles engine output file. Example:

```
.output_cycle
```

- `conn purgex`:
Purges translation caches.
Cleans up the translate caches except the `xlate/grm` cache objects, which is reloaded when the process is bounced.
- `conn pstart [-h]`:
Starts a connection.
Using the optional `-h` starts the thread with an active hold, as if it were started and then the `hold` command was issued.
- `conn pstop`:
Stops a connection.
- `prestart`:
Restarts *thread_name* in *process* after *delay interval*.
This example is the delay interval for restarting *thread_name* in *process*:

```
-p process -c thread_name prestart delay interval
```

See [hciengine restart](#).

- `conn phold_obd`:
Holds outbound data messages.
- `conn phold_obd_reply`:
Holds outbound reply messages.
- `conn prls_obd n`:
Releases outbound data messages.
Specifying the optional count, *n*, indicates that `prls_obd 2` releases only two messages without releasing the hold on the thread. It does not matter how many are in the outbound queue. If the option is not there or is zero, then the queue hold is released as in previous releases.
The count (*n*) is the maximum number of messages to release. If the queue becomes empty when releasing the messages, then the count is reset. When another message arrives, it is not released as part of the count. An empty queue clears the counting.
The count is the maximum number of messages that are released if there are not enough messages available to release. After the queue is empty, the count supplied is no longer active and the thread outbound data hold stops all messages from being processed.

- `conn prls_obd_reply`:
Releases outbound reply messages.
- `conn reindex`:
Reloads Tcl auto-load indices.
- `conn reload procname`:
Reloads Tcl auto-load procedures.
- `resend`:
Resends messages. Example:

```
conn resend \ ib [dest_thread]|ob_data|reply priority
msgfile_name len10|nl|eof [meta_file_name]
```

The optional `meta_file_name` argument to `resend` permits metadata resending.

- `meta_file_name` is an optional argument. If this argument is not supplied, then the command works the same as version 6.1.
- `msgfile_name` is the temp file that stores the message contents, which is unchanged. A new temp file is used to store edited metadata. This is referenced by `meta_file_name`. The new temp file is deleted after the `resend` is complete.
- `runnow_schedule`:
If a thread has advanced scheduling that was configured through the Network Monitor, then you can also use this command to run the scheduled task. Example:

```
hccmd -p proc_name -c "conn_name runnow_schedule"
```

Sticky hold

This is provided for the GUI to support the sticky hold option. The thread is started in the process with the last hold status:

```
hccmd -p process -c "thread_name pstart [h | H] [s | S]"
```

`-s` starts the engine with the sticky hold setting.

Specifying the destination threads when resending messages

The uniform destinations for all resent messages can be specified on the Network Monitor and SMAT resend dialog boxes. For the reply type, multiple destinations are not supported.

For other editable metadata, they can be the same as the destinations. Destinations in the metadata file are overridden by `dest` from the command line.

Note: When specifying the destination thread for the `resend`, inter-site threads are not supported.

On the command line, `resend` takes `dest_for_ib` as the option between location and type:

```
resend ib_pre_tps | ib_post_tps | ob_pre_tps | ob_post_tps [dest_for_ib]
data|reply priority filename len10|nl|eof
[metafilename]
```

Examples

Examples include:

- `hcicmd> conn_6 resend ib_pre_tps {conn_1 conn_3} data 5120 d:\\-A.txt nl`

This places two messages from file `d:\\-A.txt` into the `ib_pre_tps` data queue.

- `hcicmd> conn_6 resend ib_pre_tps {conn_1 conn_3} datc 5120 d:\\-A.txt nl`

Response: Type must be data or reply

```
resend ib_pre_tps|ib_post_tps|ob_pre_tps|ob_post_tps [dest_for_ib] data|reply pri
filename len10|nl|eof [metafilename]
```

- `hcicmd> conn_6 resend ib_pre_tps {conn_1 conn_3} datc 5120 d:\\-A.txt nlx`

Response: Incorrect options: Style `len10|nl|eof` or Type `data|reply`

```
resend ib_pre_tps|ib_post_tps|ob_pre_tps|ob_post_tps [dest_for_ib] data|reply pri
filename len10|nl|eof [metafilename]
```

resend_db

This command reads messages from a temporary, and optionally encrypted, database and sends them forward.

- The IDE and Global Monitor use `resend_db` when using a database.
- `resend` is used for file-based SMAT.

`-E` and `-D` are for the database option. The other options are the same as `resend`.

Note: "Base site" is the site on which the newly created site is based. For example, for `hcisiteinit.pl` this is the site name passed with the `-c` option, or `siteProto` without the `-c` option. For `hcirootcopy.pl`, it refers to individual sites in the site list that are passed by the `-s` or `-l` options.

Usage:

```
resend_db ib_pre_tps|ib_post_tps|ob_pre_tps|ob_post_tps filename
[-e msg_encoding] [-E [0|1]] [-D [sqlite]]
```

- Option: `-E`

Scope:

- + `hcisiteinit.pl`
- + `hcicreatesite.pl`

```
+ hcirootcopy.pl
+ hcirsiterestore.pl
* resend_db
```

Description: SMAT is encrypted (=1), or not encrypted(=0).

- If only -E is provided, without the values 0 or 1, then assume encrypted (=1).
- Encryption key is read from \$HCISITE.
- + indicates that values are stored in SMAT encrypted configuration in `siteInfo` of new site essentially overriding the value in base site.
- * Indicates that input files are encrypted or not. Default is 0.

- Option : -D

Scope:

```
+ hcisiteinit.pl
+ hcicreatesite.pl
+ hcirootcopy.pl
+ hcirsiterestore.pl
* resend_db
```

Description: The driver is used to read/write SMAT.

Acceptable values are:

- `file`: SMAT is stored in `.msg` and `.idx` files.
- `sqlite`: SMAT is stored in `sqlite` database with the `.smatdb` file extension.
- If only -D is provided without the value, then assume "sqlite".
- -E=1 and -D=file are invalid combinations.
- + indicates that values are stored in SMAT driver configuration in `siteInfo` of the new site. This essentially overrides the value in base site.
- * indicates that driver to use for input files. For `resend_db`, -D is implied.

resend_errordb

```
resend_errordb mid [file_name len10|nl|eof]
```

Resends messages from the error database.

- `mid` is the message ID in the error database.
- `file_name` is the name of the file to be used to resend as message content.

The file name is optional and only necessary when the message content is different. If the message content is the same, then the message content from the error database is reused.

When the file name is specified, the message data is replaced with the data from the file.

- `len10|nl|eof` are the ways to read the file. This is necessary only when a file name is specified.

Example

To resend a message in the error database with the message ID [0.0.3659], the command is:

```
hcicmd -p process_name -c ". resend_errordb [0.0.3659]"
```

You can also use:

```
hcicmd -p process_name -c ". resend_errordb 3695"
```

To resend a message with modified content, save the modified content in the file `myMessage.txt` and then resend it, the command is:

```
hcicmd -p process_name -c ". resend_errordb 3695 myMessage.txt eof"
```

When Message is resent, it is re-queued.

When the message is re-queued, the message is removed from the error database. The re-queued message is given a new MID, but its Source MID is the original message's source MID.

Resending with modified metadata

SMAT handles resending using the `resend` command through `hcicmd`.

```
hcicmd -p process_name -c "thread_name resend context  
type priority filename format"
```

These optional arguments to `resend` permit metadata resending:

- *metafilename* is an optional argument. If this argument is not supplied, then the command works the same as version 5.7.
- *msgfilename* is the temp file that stores the message contents, which is unchanged. A new temp file is used to store edited metadata. This is referenced by *metafilename*. The new temp file is deleted after the resend is complete.

Resending involves two temp files. Correct functioning depends on the number of messages in the message temp file matching the number of metadata entries in the metadata temp file. To assure that corrupt data is not sent in the case of a mismatch, `resend` counts the number of messages and corresponding metadata. It does this in both files before running the resend. No messages are resent in the case of a mismatch.

Message forwarding options

Message forwarding options include:

- This option takes two arguments:

```
resend_errordb -m meta_file_name
```

This resends from the error database, but with modified metadata from a file. This file takes the same format as the SMAT metadata temp file.

The message IDs that are used in the resend come from the metadata file. When using metadata, `resend_errordb` also supports resending multiple messages.

- This option takes four arguments:

```
resend_errordb -m meta_file_name msgfile_name len10|nl|eof
```

This performs an error database resend using both message data and metadata from files. As with `resend`, if the number of messages in both files do not match, the command returns an error without resending any messages.

If two or four arguments are used, then the first option must be `-m` or the command returns an error.

Because `resend_errordb` takes from one to four arguments, the full form that is called from `hcicmd` is:

```
hcicmd -p process_name -c ". resend_errordb mid"
hcicmd -p process_name> -c ". resend_errordb -m meta_file_name"
hcicmd -p process_name -c ". resend_errordb mid msgfile_name len10|nl|eof"
hcicmd -p process_name -c ". resend_errordb -m meta_file_name
msgfile_name len10|nl|eof"
```

- `-n metadata_file_name` option:

`hcidbdump` enables dumping metadata from the error database into a file of the necessary format. The option for this is `-n metadata_file_name`.

```
hcidbdump -n metadata_file_name -e -m mid output_file_name
```

For example:

```
hcidbdump -n meta.txt -e -m 0.0.1:0.0.9999 out.txt
```

This generates the metadata file. This is `meta.txt` in the example. You can modify and use this as the second argument to `resend_errordb`.

- Message encoding option:

The message encoding option permits you to specify the encoding for the data file. If there is no encoding specified, then the API keeps the old behavior.

If you only specify an MID to resend, then it is unnecessary to specify the encoding. The engine uses UTF-8 to do the resend.

This command resends the message content in the specified format into the specified process. The run `resend` command line is:

```
hcicmd -p process_name -c ". resend_errordb mid msgContent len10|nl|eof
[-e msg_encoding]"
```

- `save_start` starts saving protocol messages:

```
conn save_start {in|out} file
```

- `save_stop` stops saving protocol messages:

```
conn save_stop {in|out}
```

- `save_cycle` cycles the save message file:

```
conn save_cycle {in|out}
```

- `show` displays information about the alerts that are currently loaded by `hcimonitor`.

```
show type
```

Where *type* is one of:

- `all` shows all loaded alerts.
- `alert name` shows the alert named *name*.
- `group groupname` shows alerts belonging to group *groupname*.
- `config file_name` shows alerts loaded from the configuration file *file_name*. For example, `default.alert`.
- `disabled` shows alerts currently disabled by `hcicmd` commands.
- `enabled` shows enabled alerts that have not been disabled by `hcicmd` commands.
- `window` shows alerts currently in their active window.
- `window_not` shows alerts currently not in their active window.
- `active` shows alerts that are armed or fired.
- `THREADCTL` controls the thread remotely. Uppercase is required.

```
THREADCTL process_name threadname
```

- `xhold_post` holds the post-xlate queue.

```
xlt xhold_post conn
```

- `xrel_post` releases the post-xlate queue.

```
xlt xrel_post conn
```

To use these commands in a non-interactive manner, use `-c` with `hcicmd`.

Resending examples

To stop the thread named `test1` in the running engine process *testsite*:

```
hcicmd -p testsite -c "test1 pstop"
```

To resend an outbound data message to the same thread in the same process:

- Data message is named `/tmp/resend_test1`.
- Priority is 5120. This is the default.
- File is `nl`. This is newline terminated.

For example:

```
hcicmd -p testsite -c "test1 resend ob data 5120/tmp/resend nl"
```


To see a list of active client connection IDs on the `tcp_multi_server` connection:

```
hcicmd -p multi -c "tcp_multi_server client_list"
```

Message forwarding commands

`forward start`. To start forwarding data messages to the thread that is specified by the `Forward Thread` Network Configurator entry, use:

```
hcicmd -fwd_start
```

`forward stop`. To stop forwarding data messages:

```
hcicmd -fwd_stop
```

`forward reply start`. To start forwarding reply messages to the thread that is specified by the **Forward Thread** field in Network Configurator:

```
hcicmd -fwdr_start
```

`forward reply stop`. To stop forwarding reply messages:

```
hcicmd -fwdr_stop
```

Advanced scheduling through hcicmd

Because scheduled events are recurring events, certain commands are provided to check the status and to disable/enable individual events or entire schedules.

After `hcicmd` is started, the thread name is specified. This is followed on the same line with the command, and then followed by any arguments that are required for the command.

These commands include:

- `disable_event`
- `enable_event`
- `disable_schedule`
- `enable_schedule`
- `schedule_status`
- `event_status`
- `runnow_schedule`

Event-specific commands require the event number as an argument. For example, event number `1` in a thread called `from_hci_adt` requires disabling. The command that is specified at the `hcicmd` prompt, or sent to `hcicmd` from Network Monitor, is:

```
from_hci_adt disable_event 1
```

The string returned from this command is `Event 1 disabled`. The return from `schedule_status` is `UP` or `DOWN`. If the schedule is `UP`, then the string also contains the number of each enabled event. For example, `UP 1 5 8`.

Using `runnow_schedule` runs the scheduled task immediately.

hconndump

This command shows threads and processes information from the Netconfig file within the site.

```
hconndump [-v] {-p process|conn_name,,}
```

- `-v` specifies verbose dump.
- `-p process` dumps all connections in the indicated process.

You can also use:

`conn_name...`, which names all the connections to dump.

hconnstatus

This command shows operational status of threads and processes. The default output is a summary of all processes and connections.

This command does not function if the Monitor Daemon is not operational.

```
hconnstatus [{-p process| -c connection}] [-s {P|T}] [-i] [-h]
```

- `-p process` displays all connections in *process*.
- `-c connection` displays the status of the indicated connection.
- `-s` sorts by `P` (process) or `T` (thread).
The default is sort by process, then by thread.
- `-i` displays the entire process name and connection name without truncation.
- `-h` displays help.

hcicreateformatdb

This command is automatically called when saving a variant through the HL7 GUI or deploying a BOX with HL7 variants included.

If the variant is changed outside the GUI, then you must use `hcicreateformatdb` to add the variant to the Auto-Complete list.

```
hcicreateformatdb {-s sitename | -a}
```

- `-s sitename` is the site in which to generate the format database file. This can be one site, or a comma-separated list of sites.
- If no site is given, then the `$HCISITEDIR` environment variable must be defined to use as the default option.
- `-a` generates the format database file for all sites in the `server.ini` file.

hccreatesite

This command creates a new site. You must specify the root and site name.

```
hccreatesite [-c site] [-p path] [-E SMATDBFlag] [-D type] [-K SMATDBKey]
[-I InternalDBFlag] [-M InternalDBKey] [-R ErrDBFlag] [-N ErrDBKey] root newsite
```

- `-c site` is the site to be cloned by *newsite*.
 - If `-c site` is used and no flags are used, then database encryption stays consistent with the configuration in the template site.
 - If `-c site` is not used and no flags are used, then the databases are encrypted by default and use the default key.
- `-p path` is the target path where *newsite* is saved.
A symbolic link `$HCIRoot/newsite` is created linking to `<path>/newsite` after site creation. This is available only in NTFS on Windows. The target path must be R/W to the user.
- `-E SMATDBFlag` encrypts/decrypts SMAT.
 - `1` indicates encrypted. This is the default.
 - `0` indicates unencrypted.
- `-D type` is the driver type that is used to read/write SMAT. Acceptable values are `file` and `sqlite`. When `-D type` is `sqlite`, the `sqlite` database retains the same behavior as the internal and error databases.
- `-K SMATDBKey` is the encryption key of the SMAT database.
- `-I InternalDBFlag` is the internal database. This flag has a higher priority than the encryption setting in the template site specified by `-c site`.
 - `1` indicates encrypted. This is the default.
 - `0` indicates unencrypted.
- `-M InternalDBKey` is the encryption key of the internal database.
- `-R ErrDBFlag` is the error database. This flag has a higher priority than the encryption setting in the template site specified by `-c site`.
 - `1` indicates encrypted. This is the default.
 - `0` indicates unencrypted.
- `-N ErrDBKey` is the encryption key of the error database.
- *root* is the name of the root to create the new site under *newsite*.
- *newsite* is the name of the new site to create. This must not currently exist.

hcidbcheck

Along with `hcidbdump -R`, `hcidbcheck` is used to aid in recovering the recovery database outside the engine.

```
hcidbcheck -r [-v]
```

- `-r` checks the recovery database.
- `-v` is verbose (detailed).

hcidbconvert

This command converts messages in the current site's Raima database to the SQLite database.

```
hcidbconvert [-p path] [-v] -e
```

- `-p path` is the file path where the exported Raima database files are located. If this option is not specified, then the `exec/databases` folder of the current site is used.
- `-v` is verbose mode.
- `-e` converts messages from error database.

Converting the error database

Conversion steps for the error database:

- 1 `setsite $srcSiteName` to the site from which data is to be converted.
- 2 Run `dbexp elog` to export data from database `elog` to an ASCII text file.
Files `elogctxmsg.txt`, `elogmsg.txt` and `elogmsg2000.txt` are generated under the current directory.
- 3 `setsite $destSiteName` to the site to which the data is to be converted.
- 4 Copy the exported Raima ASCII text files to the `exec/databases` folder of the destination site.
- 5 Run `hcidbconvert -e` to convert error database data.

hcidbcrypt

The error database is in a SQLite database, where you can edit, resend, and search the error database. A user interface similar to SMAT is available to manage the database.

This command is provided for users to change the error and internal database passwords.

```
hcidbcrypt {-e|-I} [-d] [-p password]
```

Both the error and internal database are supported for encrypting/decrypting with a user-defined password.

- `-e` specifies encryption/decryption of the error database.
- `-I` specifies encryption/decryption of the internal databases. For example, the recovery database..

- `-d` specifies to do decryption, when it is given; otherwise, encryption.
- `-p password` specifies the encryption key.
If this option is not specified, then the default encryption key is used.
- `-q` indicates quiet, or no output except errors.

To change the error database password:

- Decrypt the error database or internal database using the old password:

```
hcidbencrypt {-e|-I} -d -p oldpassword
```

- Encrypt the error database or internal database with the new password:

```
hcidbencrypt {-e|-I} -p newpassword
```

If the `-p` option is not used in the command, then the engine uses the default password to encrypt/decrypt the database.

siteSecurityInfo keys

These keys in `siteSecurityInfo` are updated based on the encryption/decryption option:

- `dbencryption`
The default is 1: Raima databases are encrypted.
- `internaldbkey`
The default is "": Use the default Raima database key for encryption.
- `errordbencryption`
The default is 1: Error database is encrypted.
- `errordbkey`
The default is "": Use the default error database key for encryption.

`siteSecurityInfo` is encrypted. The keys that are saved in this file are plain text.

hcidbdefrag

This command reclaims disk space.

Note: If you have a defined database alert, then you must shut down the MonitorD process and Lock Manager before running this command. For example, a defined database alert can be message archiving or error database. Running `hcidbdefrag` with these alerts produces an error.

Whenever a record is deleted from the database, it is marked for reuse and put on the delete chain. It is not physically removed from the file. These free (deleted) records occupy a substantial amount of disk space, especially if a database has had numerous records inserted with many marked as deleted.

This utility moves all valid records to the front of a data file and all deleted slots to the end of the data file. The data file is then truncated to reclaim disk space.

```
hcidbdefrag [-A] [-r] [-e] [-i] [-m [-s threadname]] [-t] [-q] [-v]
```

- `-A` defrags all databases.
- `-r` defrags the recovery database.
- `-e` defrags the error database.
- `-i` defrags the ICL registry database.
- `-m` defrags the Message Archive database cache.
- `-s threadname` *threadname*.
- `-t` defrags the message tracing database cache.
- `-q` suppresses all output, except errors.
- `-v` specifies verbose operation. This shows detailed output of the operations performed on the database. defrags the message archive database cache for

hcidbdump

This command accesses the same functions as the Database Administrator GUI. This supports a regex search on message content and dump error context.

Note: Do not terminate `hcidbdump`! Termination before it is finished corrupts the database.

You can use one of these formats:

```
hcidbdump [-U userid] [-v] -B [tarfile]
```

```
hcidbdump {-r|-e} [-U userid] [-l] [-L [-x]] [-v] [-D] [-i] [-f source_system]
[-o owner_system] [-d dest_system] [-t type] [-s state]
[-S time] [-E time] [-M map] [-O time] [-a] [-b [eof]]
[-F] [-n metadata_filename] [-m mid] [-c] [-C] [-R] [-w searchExp]
[output_file]
```

- `-B` performs database backup.
- *tarfile* specifies the file name where the database backup is written. If no *tarfile* is specified, then tar is written to `$DBFPATH/cldb.tar`.
- `-r` saves message from recovery database.
- `-e` saves message from error database.
- `-Uuserid` specifies the user ID to access database.
- `-l` specifies long form output. This is ignored when *output_file* is specified.
- `-L` specifies ultra-long form output. Overrides `-l`.
- `-x` gives an expanded view of flags. This is ignored unless using `-L`.
- `-v` specifies verbose operation.
- `-D` deletes messages after processing.
- `-i` shows the entire process and thread name without truncation.
- `-f source_system` selects messages with a specified source.
- `-o owner_system` selects messages with a specified owner.

- `-d dest_system` selects messages with a specified destination.
- `-t type` selects messages with a specified type:
 - `D` specifies data.
 - `R` specifies reply.
 - `U` specifies unknown.
- `-s state` selects messages with a specified state.
- `-S time` selects messages at or after a specified date/time.
- `-E time` selects messages at or before a specified date/time.
- `-M map` applies the map to each message. Uses the `msgmapdata` command.
- `-O time` orders output by ascending date. Time flags are:
 - `i` specifies inbound arrival time.
 - `o` specifies outbound time.
 - `r` specifies recovery time.
 - `x` specifies xlate start time.
- `-a` appends data to output file.
- `-b [eof]` dumps message to output file in length-encoded or end-of-file format.
 - `-b` dumps in length-encoded format.
 - `-b [eof]` dumps in end-of-file format. Argument [*eof*] is optional.

Note: You must provide an output file name for `-b [eof]`; otherwise, an error message results.
- `-F` deletes sent messages without prompting for confirmation.
- `-n metadata_filename` enables metadata to be dumped from the error database into a file of the necessary format.
- `-m mid` messages with specified message IDs:
 - `x.y.z` specifies only this message.
 - `x.y.z1:x.y.z2` specifies `x.y` messages `z1` through `z2`.
 - `-m` overrides all other restrictions.

Note: `-m` does not work with `-o`.
- `-c` displays context from error database.
- `-c` prints only the message count.
- `-R` is recovery mode for the recovery database.
- `-w searchExp` supports regular expression search on message content. The search is case-sensitive, and works only for the error database.
- `output_file` specifies the file name where messages are written. If no output file is specified, then message data is written to `stdout`.

All specified constraints are ANDed together.

Recovery of threads outside the engine

A flag in `hcidbdump` forces all corrupt messages from the recovery to the error database to state 500 for all threads in a site.

```
hcidbdump -r -U userid
```

`-U userid` permits the user to give a new user ID for accessing the database with `hcidbdump`. With this usage, you can move the corrupt messages to the error database. The owner of the error database messages is *userid*.

This command usage scans messages in the recovery database with user *userid*. If corrupt messages are found during scanning, then `hcidbdump` automatically recovers with user *userid*.

When running this command on a corrupt database, these should be observed:

- Back up the database and ensure no engine process is running on the site.
- Because the database is corrupt, it may encounter a raima system error (-900 ~ -966) during the automatic recovery. In this case, `hcidbdump` panics to prevent additional errors accumulating and eventually corrupting the database again.
- If the error of the corrupt message is `Read msg data chain but mid not exist; err 2`, then run the `keybuild rlog` command first to rebuild the key file for the recovery database. After that, run `hcidbdump -r -U userid` again.

Note: If there is a -944 error when running `keybuild`, then delete the `$DBTAF/rdm.taf` file. Then run `keybuild rlog` again.

Transaction logging (trlogging) option

This option is related to recovery in that it helps to reduce potential corruption.

The transaction logging option is used for transaction logging and recovery. When this option is on, RDM Embedded automatically logs all transactions.

In this way, if transactions abnormally terminate, automatic recovery is accomplished based on the log files. Otherwise, no transaction logging is performed and all updates are made directly to the database.

The default value for this option is "Off." This option can only be changed if all databases are closed.

Note: Enabling the `trlogging` option introduces some overheads which could affect performance.

Turning on transaction logging

To turn on this option:

- 1 Shut down all processes and the Monitor Daemon.
- 2 Open the `rdm.ini` file from the `$HCISITEDIR/exec/databases` directory.
- 3 Change the option `trlogging` under the `[rdm]` section to 1:

```
[rdm]
...
trlogging=1
...
```

- 4 Save the `rdm.ini` file.
- 5 Restart all processes and the Monitor Daemon.

Completing the recovery

- 1 Back up the Raima embedded database files that are located at `$HCISITE/exec/databases/*.*`. Back them up into a newly created folder, for example, `rdmebackup`.
- 2 Stop all processes in the current site.
- 3 Run the `hcidbcheck -r` command. This command first rebuilds the key file of the recovery database. Then, it checks the DATA file consistency to see if there are still errors in the recovery database. If it returns "0" errors found, then the database is recovered; otherwise, go to the next step.
- 4 Run `hcidbdump -r -R` to transfer the corrupt messages to the error database.
- 5 Start the engine processes.

If the engine starts successfully without Raima system errors (-9XX), then the recovery is complete; otherwise, you must initialize the recovery database because the corrupt database cannot be recovered.

See [hcidbcheck](#).

hcidbinit

This command reinitializes the databases. This initializes the SQLite error database file, not the Raima error database files.

```
hcidbinit [-A] [-r] [-e] [-i] [-m [-s threadName]] [-t] [-C] [-f]
```

- `-A` initializes all databases.
- `-r` initializes the recovery database.
- `-e` initializes the error database.
- `-i` initializes the ICL registry database.
- `-m` initializes the Message Archiving database cache.
- `-s` initializes a Message Archive database cache for a thread.
- `-t` initializes the Message Tracing database cache.
- `-c` cleans control files.
- `-f` forces running and skips query.

Note: If all the databases are initialized with the `-A` flag, then also use the `-c` option. Using the `-A` option continues to initialize all databases, including the message archive database.

hcidbprotocoltest

This command tests the specified database protocol thread.

```
hcidbprotocoltest [-a] [-d n] [-e enc] [-w searchExp]
[-s outfile] [-r] [-f format] thread [infile]
```

- `-a` processes all records in the file.
- `-d n` shows data at detail level *n*.
- `-e enc` encodes from *enc* to UTF-8.

- `-w searchExp` prints only the lines of the output-matched search expression. The search is case-sensitive.
- `-s outfile` is the newline-terminated data file for queried database schema messages. This is valid only for the Inbound dbprotocol thread.
- `-r` runs the SQL statements to insert/update the data in the database.
- `-f format` specifies the input file format.
 - `-f len10` specifies 10-byte length-encoded.
 - `-f nl` specifies newline terminated. This is the default.
 - `-f eof` specifies end-of-file terminated.
- `thread` is the source thread to simulate.
- `infile` is the data file with which to test.

hcidbschematest

This command parses DB schemas in the SMAT tool.

```
hcidbschematest [-a] [-e encoding] [-d n] [-f format]
[-w search expression] dbschema file
```

- `-a` processes all records in the file.
- `-e encoding` encodes from enc to UTF-8.
- `-d n` shows data at detail level *n*. The detail level range is 0-4, where 0 is raw, unparsed data and 4 is the most detailed.
- `-f format` specifies the input file format.
 - `-f len10` specifies 10-byte length-encoded.
 - `-f nl` specifies newline terminated. This is the default.
 - `-f eof` specifies end-of-file terminated.

`-w search expression` specifies the search string or regular expression for which to search. The search is case-sensitive.

`dbschema` is the name of the DB schema to test.

`file` specifies the data file with which to test.

hcidbsetvers

This command sets user access to the database. This gets/sets the error database versions from the SQLite error database file, not the Raima error database files.

```
hcidbsetvers [-o] [-f] [-q] [-u userid] [-v]
```

- `-o` specifies one user database access.
Note: Do not use this method! Use the Lock Manager to restrict user access.
- `-f` forces action; skips user query.
- `-q` queries the version only; no update.

- `-u userid` specifies user ID for database access.
- `-v` specifies verbose operation.

hcidcmehoscu

You can use this syntax to run `hcidcmehoscu.exe` from the command line:

```
hcidcmehoscu [peer] [port] [-aet | -aetitle] [-aec | -call]
[-ta | -timeout] [-ll | -loglevel]
```

- `peer` is the host name of the DICOM peer. The default is *hostname*.
- `port` is the TCP/IP port number of the peer. The default is 11112.
- `-aet | -aetitle` calls the AE (Application Entity) title. The default is CL-ECHO-SCU (Service Class User).
- `-aec | -call` calls the AE title of the peer. The default is ANY-SCP (Service Class Provider).
- `-ta | -timeout` is the time-out for DIMSE (DICOM Message Service Element) messaging. The default is 60 seconds.
- `-ll | -loglevel` is the log level:
 - debug
 - disabled
 - error
 - fatal
 - info
 - trace
 - warn

hcidicomdefgen

This command generates site-level LDL definition files based on an input DICOM message.

```
hcidicomdefgen -V version -v variant -s service_name
-c transfer_syntax input_message
```

- `-V version` is the DICOM version.
- `-v variant` is the DICOM variant.
- `-s service_name` is the DICOM service name.
For example: `C_STORE_RQ`, `C_FIND_RSP`,...
- `-c transfer_syntax` is the transfer syntax of the DICOM message.
- `input_message` is the input DICOM message.

This command:

- Creates a variant under `$HCISITEDIR/formats/ldl/version/`.
- Creates a segment named `service_name_dataset`, and generates the content according to the input message parsing result.
- Creates `service_name_MSG` under `$HCISITEDIR/formats/ldl/version/variant/messages`, which contains:

- `service_name_DIMSE`: This should already be defined under `$HCIR00T/formats/ldl/version/variant/segments`. This command does not generate the definition file for the DIMSE header part.
- `service_name_dataset`: Content is generated based on the dataset part of the input DICOM message. This file is located under `$HCISITEDIR/formats/ldl/version/variant/segments`.

Example:

```
hcidicomdefgen -V 2014 -v test -s C_STORE_RQ -c "{TransferSyntax 1.2.840.10008.1.2.4.91}"
c_store.dat
```

Reusing the LDL file

You can create a variant using `hcidicomdefgen`. For example:

```
hcidicomdefgen -V version -v variant -s service_name -c transfersyntax input_message
```

This creates a variant specifying the `service_name` `C_STORE` and `transfersyntax` `1.2.840.1008.1.2` [Little Endian].

With this usage, you can receive `C_STORE_RQ` messages that have a different transfer syntax for Big Endian.

You can also reuse the same variant to handle different transfer syntaxes.

For example, you can create two variants first, then combine them into one variant.

In this example, there are two `C_STORE_RQ` messages:

- `message1` uses `transfersyntax 1.2.840.1008.1.2`.
- `message2` uses `transfersyntax 1.2.840.1008.1.2.4.91`.

The command is:

```
hcidicomdefgen -V 2014 -v test1 -s C_STORE_RQ -c "{TransferSyntax 1.2.840.10008.1.2}" message1
hcidicomdefgen -V 2014 -v test2 -s C_STORE_RQ -c "{TransferSyntax 1.2.840.10008.1.2.4.91}" message2
```

This generates variant `test1` for `message1` and variant `test2` for `message2`.

To use variant `test1` for `message1` and `message2`, merge the directory `test2` to directory `test1`.

hciedifactest

This command tests UN/EDIFACT configurations against actual data.

```
hciedifactest [-i] [-e encoding] [-n] [-a] [-d n]
[-f format] [-V version] [-v variant]
[-t test] [-F fieldname] file
[-w search expression]
```

- `-i` shows field addresses using index notation.
- `-e encoding` is the encoding name for the data file.
- `-n` shows the field names.
- `-a` processes all records in the file.

- `-d n` shows data at detail level *n*. The detail level range is 0-4, where 0 is raw, unparsed data and 4 is the most detailed.
- `-f format` specifies the file format:
 - `-f len10` specifies 10-byte length-encoded.
 - `-f nl` specifies newline terminated. This is the default.
 - `-f eof` specifies end-of-file terminated.
- `-f format` specifies the UN/EDIFACT version.
- `-v variant` specifies the UN/EDIFACT variant.
- `-t test` specifies the test to run:
 - `-t numbers` tests *Nn* number conversions.
 - `-t parse` test parses a message. This is the default.
- `-F fieldname` specifies the field name only.
- `file` specifies the data file with which to test.
- `-w search expression` specifies the search string or regular expression for which to search.
- `-p sepChars` are the separator characters used for parsing or encoding. This needs to be in a keyed list format.

hcienginerestart

This command restarts engine processes, threads, and MonitorD. This supports the GUI for restarting an engine process, thread, and MonitorD.

You can use one of these commands:

```
hcienginerestart -d delay interval -p process [,process...] [-n]
```

```
hcienginerestart -d delay interval -p process [,process...]
[-h host] [-s thread [, thread...]] --
allElse [-n]
```

- `-d delay interval` is the unit for delay interval is seconds. The default value is 30s, and is defined in the GUI at the site-level configuration. For a different time-out, use the command-line directly.
- `-p process [,process]` restarts the indicated processes.
- `-h host` is the name of the remote host to run the engine on.
- `-s thread` is a comma-separated list of threads to restart.
- `--` separates the script args from engine args.
- `allElse` specifies all other arguments are passed to the engine.
- `-n` specifies to not run the engine in a service on Windows.

If `hcienginerestart` is called when the process is not running, then the `hcicmd` invocation to stop fails.

If it can verify that the process is not running after the delay interval has elapsed, then it attempts to start it with the remaining arguments.

The `-p` argument can be a process list.

If the process cannot be stopped, then it returns an error after the delay interval has elapsed.

For example, to restart a process:

```
hciengine restart -d delay interval -p process
```

Threads

```
hcicmd -p process -c "thread_name prestart delay interval"
```

- If `hcicmd` is invoked with `thread_name prestart` when the process is not running, then it fails to connect. In this case, you can invoke `hciengine run` with the thread start args. For example:

```
hcicmd -p process -c thread name prestart  
delay interval
```

- If `hcicmd` is invoked when the process is running but the indicated thread is not, then it starts the thread after the delay interval has elapsed.
- If `hcicmd` is invoked when the process is running but the indicated thread has not stopped, then it returns an error. This is returned after the delay interval has elapsed.

Site daemons

```
hcisitectl [-f] [-h host] [{ -K | -k daemon }] [{ -S | -s daemon }]  
[{ -R | -r daemon} -d delay interval] [-u #users]  
[-A args]
```

For site daemons it restarts when the `-R` or `-r` argument is used.

- If `hcisitectl` is called this way when the daemon is not running, then the invocation to stop fails. If it verifies that the process is not running after the delay interval has elapsed, then it attempts to start it.
- If it is unable to stop the daemon, then it returns an error after the delay interval has elapsed.
- If using `-R` or `-r`, then the `-d` argument is required.
- For example:
 - To restart the MonitorD:

```
hcisitectl -r m -d delay interval
```

- To restart the Lock Manger:

```
hcisitectl -r l -d delay interval
```

- To restart all daemons:

```
hcisitectl -R -d delay interval
```

The unit for *delay interval* is seconds. The default value is 30s, and is defined in the GUI at the site-level configuration. For a different time-out, you can change *delay interval*.

hciengerun

This command starts the engine, runs the monitor, and cleans up.

```
hciengerun -p process[,process...] [-h host]
[-s thread[,thread...]] [-o thread[,thread...]]
[-x-x thread[,thread...]] [-n] -- allElse
```

- `-p process` is the name of the process to start.
- `-h host` is the name of the remote host on which to run the engine.
- `-s thread` is a comma-separated list of threads to start. The use of this flag over-rides the auto-start setting for the named threads and starts them upon process startup.
- `-o thread` is a comma-separated list of threads to start with OB hold. The use of this flag over-rides the auto-start setting for the named threads and starts them with OB hold.
- `-x thread` is a comma-separated list of threads to start with OB hold sticky. The use of this flag over-rides the auto-start and `-o` setting for the named threads, and starts them with OB hold sticky.
- `-n` specifies to not run the engine in a service on NT.
- `--` separates script args from engine args.
- `allElse` specifies all other arguments are passed to the engine.

Note: All flags directed at this script must precede arguments directed to the engine. This permits the script to remain ignorant of all the options that the engine supports.

Starting the engine without starting threads

Even if the threads are set to automatically run, you can configure `hciengerun` to start the engine but not any threads. The threads can be started at any later time.

To do this, add the `"-0"` (minus zero) parameter after `--` (minus minus) on the command line. The result is as if none of the threads in the process are set to auto-start.

This feature works only when `hciengerun` is used to start the engine. All flags directed at this script must precede arguments aimed at the engine. This permits this script to remain ignorant of all the options that the engine supports.

hcienginestop

This command stops an engine process.

```
hcienginestop -p process [,process...]
[-h host]
```

- `-p process` stops the indicated processes.
- `-h host` specifies the name of the remote host on which to run.

hcieols

This command lists all engine output aliases.

hcifrlconvert

This command converts the fixed-record configuration file between versions.

```
hcifrlconvert [filename]
```

hcifrls

This command lists the FRL configuration files in a site.

You can use one of these commands:

```
hcifrls [-lp]
```

```
hcifrls [-l] [-p]
```

- `-l` shows the long listing.
- `-p` shows file prologues.

hcifrltest

This command tests fixed-record configurations against actual data.

```
hcifrltest [-e encoding] [-a] [-d n] [-f format] frl  
file [-w search expression]
```

- `-e encoding` is the encoding name for the data file.
- `-a` processes all records in the file.
- `-d n` shows data at detail level *n*. The detail level range is 0-4, where 0 is raw, unparsed data and 4 is the most detailed.
- `-f format` specifies the file format:
 - `-f len10` specifies 10-byte length-encoded.
 - `-f nl` specifies newline terminated. This is the default.
 - `-f eof` specifies end-of-file terminated.
- *frl* specifies the FRL name to test.
- *file* specifies the data file with which to test.
- `-w search expression` specifies the search string or regular expression for which to search.

hcigencerts

This command is used in setting up security server default accounts.

User certificates are issued by the CA certificate with the default password.

```
hcigencerts -cacertpath certpath -capubcert pubkeyname -caprivkey prikeyname
-capass capassword -pass certpassword -template filepath
[-l logfile ] [-h|-help]
```

- `-cacertpath certpath` is the path of the CA certificate.
- `-capubcert pubkeyname` is the file name of the CA public key.
- `-caprivkey prikeyname` is the file name of the CA private key.
- `-capass capassword` is the password of the CA certificate.
- `-pass certpassword` is the default password of the new user certificates.
- `-template filepath` is the predefined `xml` file with full path.
- `-l logfile logfile` is the file name of the log.
- `-h|-help` prints this help message.

hcigetnextport

This command locates the next available port, starting from the lowest possible port. If the range is not specified by the `-l` or `-h` arguments, then the default range is from 1024 to 49151.

```
hcigetnextport {-a|-s} [-l min_port] [-h max_port] [-r hciroot1
[,hciroot2...] [-i site1[,site2...]]
```

- `-a` excludes those ports that are actively in use in the system.
- `-s` excludes those ports that have been configured as server ports in any of the sites under the specified root. It issues a warning if the port returned is actively in use in the system.
- `-l min_port` defines the inclusive lower limit of the search range. Port numbers lower than *min_port* are excluded.
- `-h max_port` defines the inclusive upper limit of the search range. Port numbers higher than *max_port* are excluded.
- For `-r hciroot1[,hciroot2...]`. If `-s` is selected, then this argument defines `hciroot` under which the command does the search. If not specified, then this tool uses the current root defined in the `HCIRoot` environment variable.
- For `-i site1[,site2...]`. If `-s` is selected, then this argument defines site under which the command does the search. If not specified, then this tool searches all sites.

hcigvt

This command is used to access the global variables.

```
hcigvt {add|del|get|set|show} [varname] [varvalue] [isencrypted]
```

- `add` adds the global variable to the variable table.
- `del` deletes the global variable name from the variable table.
- `get` prints the global variable value of *varname*.

- `set` sets the global variable value of *varname* to a new value.
- `show` shows all the variable names and values in local site.
- *varname* is the global variable name.
- *varvalue* is the global variable value.
- `isencrypted` determines if the global variable must be encrypted in the `ini` file.
 - `1` indicates it is encrypted.
 - `0` indicates it is not encrypted. This is the default.

For example:

```
C:\cloverleaf\cis6.3\integrator\bin>hcigvt show
Global variable testPass = passwd1, encrypt = 0
```

hcihd

This command outputs the hex dump of a file.

```
hcihd [file]
```

hcihl7test

This command defines HL7 configurations against actual data.

```
hcihl7test [-a-a] [-d n] [-e encoding] [-i] [-f format]
[-V version] [-v variant] [-t test] [-M message]
[-c drivercontrol] [-n] [-F fieldname] [-w search expression]
[-p sepChars]
```

- `-a` processes all records in the file.
- `-d n` shows data at detail level *n*. The detail level range is 0-4, where 0 is raw, unparsed data and 4 is the most detailed.
- `-e encoding` is the encoding name for the data file.
- `-i` shows field addresses using index notation.
- `-f format` specifies the file format:
 - `-f len10` specifies 10-byte length-encoded.
 - `-f nl` specifies newline terminated. This is the default.
 - `-f eof` specifies end-of-file terminated.
- `-V version` specifies the HL7 version.
- `-v variant` specifies the HL7 variant.
- `-t test` picks which test to run.
- `-c drivercontrol` is used to set the content to message metadata DRIVERCONTROL.
- `-n` shows the field names.
- `-F fieldname` specifies the field name only.
- `-w search expression` specifies the search string or regular expression for which to search.

- `-p sepChars` are the separator characters used for parsing or encoding. This needs to be in a keyed list format.

hcieprimtest

This command tests HPRIM record format configurations. This test reads, parses, and displays messages, ensuring that configurations match the input data.

```
hcieprimtest [-a] [-e encoding] [-d n] [-i] [-f format]
[-F fieldname] [-w search expression] [-p sepChars]
[-V version] [-v variant] [-t test] [-c drivercontrol]
[-n] file
```

- `-a` specifies to read the selected data file and process all the messages in it.
- `-e encoding` is the encoding name for the data file.
- `-d n` shows data at detail level *n*. The detail level range is 0-4, where 0 is raw, unparsed data and 4 is the most detailed.
- `-i` displays field addresses using index notation.
- `-f format` specifies the file format:
 - `-f len10` specifies 10-byte length-encoded.
 - `-f nl` specifies newline terminated. This is the default.
 - `-f eof` specifies end-of-file terminated.
- `-F fieldname` shows the listed field name.
- `-w search expression` specifies the search string or regular expression for which to search.
- `-p sepChars` are the separator characters used for parsing or encoding. This needs to be in a keyed list format.
- `-V version` specifies the HPRIM version.
- `-v variant` specifies the HPRIM variant.
- `-t test` picks which test to run.
- `-c drivercontrol` is used to set the content to message metadata DRIVERCONTROL.
- `-n` shows the field names.
- `file` specifies the data file with which to test.

hciehrlconvert

This command adds the {ACTIVENULLWHENNOCONTENT 1} key into any VRL/FRL segment in an HRL file.

```
hciehrlconvert [hrl ...]
```

`hrl` is the HRL file to convert.

Specifying no argument converts all HRL files under the site.

The ACTIVENULLWHENNOCONTENT key is used to change the parsing and encoding behavior on a empty VRL/FRL segment in an HRL file.

In versions earlier than 6.1, the behavior was to set the empty VRL/FRL segment as ActiveNull.

In versions 6.1 and later, the default behavior is to fill padding characters for empty VRL/FRL segments.

For example, an xlate file uses **Repeat While** on an empty VRL/FRL segment in a HRL. You can set `ACTIVENULL WHENNOCONTENT` to 1 for the VRL/FRL segment in the HRL. This is set through the GUI by selecting **Active Null** or using `hcihrlconvert` on the command line.

If you have no requirement to use HRL in this way, then do not select Active Null on the GUI. This makes the engine use the new behavior.

hcihrls

This command lists the HRL configuration files in a site.

You can use one of these commands:

```
hcihrls [-lp]
```

```
hcihrls [-l] [-p]
```

`-l` shows the long listing.

`-p` shows file prologues.

hcihrltest

This command tests HRL data against actual data.

```
hcihrltest [-a-a] [-e encoding] [-d n]  
[-f format] hrl file [-w search expression]
```

- `-a` processes all records in the file.
- `-e encoding` is the encoding name for the data file.
- `-d n` shows data at detail level *n*. The detail level has a range of 0-4, where 0 is raw, unparsed data and 4 is the most detailed.
- `-f format` specifies the file format:
 - `-f len10` specifies 10-byte length-encoded.
 - `-f nl` specifies newline terminated. This is the default.
 - `-f eof` specifies end-of-file terminated.
- *hrl* specifies the HRL name to test.
- *file* specifies the data file with which to test.
- `-w search expression` specifies the search string or regular expression for which to search.

hcijsonetest

This command verifies the JSON message according to the specified JSON format and prints the JSON message in the data file.

```
hcijsonetest [-a] [-d n] [-e encoding] [-f format]
              [-w searchExp] -p package schema data file
```

- `-a` processes all data in the file.
- `-d n` shows data at detail level *n*.
- `-e encoding` encodes from *encoding* to UTF-8.
- `-f format` specifies the file format:
 - `-f len10` specifies 10-byte length encoded.
 - `-f nl` specifies newline-terminated.
 - `-f eof` specifies end-of-file terminated. This is the default.
- `-w searchExp` indicates only the lines of output that match the search expression are printed. The search is case-sensitive.
- `-p package` is the directory containing the JSON definition file.
- `schema` schema file name of message definition.
- `data file` is the data file to test with.

hcildltest

This is a parse testing tool for the LDL file.

```
hcildltest [-a] [-d n] [-e enc] [-i] [-f format]
            [-V vers] [-v var] [-t test] [-M message]
            [-c drivercontrol] [-n] [-F fieldname] [-w searchExp]
            [-p sepChars] file
```

- `-a` processes all records in the file.
- `-d n` shows data at detail level *n*.
- `-e enc` encodes from *enc* to UTF-8.
- `-i` displays field addresses using index notation.
- `-f format` specifies the file format:
 - `-f len10` specifies 10-byte length-encoded.
 - `-f nl` specifies newline terminated. This is the default.
 - `-f eof` specifies end-of-file terminated.
- `-V vers` is the *ldl* version. This is a generic type; the default depends on the type.
- `-v var` is the *ldl* variant. This defaults to `''`.
- `-t test` is used to select which test to run:
 - `numbers` tests *Nn* number conversions.
 - `parse` test parses a message. This is the default.
- `-M message` is the message file for the DICOM message.
- `-c drivercontrol` is used to set the content to message metadata `DRIVERCONTROL`. You can input any content to set to *drivercontrol* using this option.

For `ldl` parse, `-c drivercontrol` is required, where `drivercontrol` is `{TransferSyntax***}`.

See [DICOM transfer syntax values](#).

Example:

```
hcildltest -i -f eof -V 2014 -v test -M C_MOVE
-c "{TransferSyntax 1.2.840.10008.1.2}" c_store.dat
```

- `-n` displays field names.
- `-F fieldname` displays only the field name.
- `-w searchExp` indicates only the lines of output that match the search expression are printed. This is case-sensitive.
- `-p sepChars` are the separator characters used for parsing or encoding. This should be in keyed list format.
- `file` is the data file with which to test.

hcilen10dump

This command shows length-encoded messages in a file.

```
hcilen10dump [-h] filenames
```

`-h` specifies a hex output of messages.

hcilichelp

The command line license help tool is a streamlined method to gather human-readable license information from their installations.

Command usage is:

```
hcilichelp [-i license filename] [-o report filename]
[-c CSC root] [-a]
```

- `-i license filename` is an optional license file name for the license file used for the report. If no license file is supplied, then the report uses the current root.
- `-o report filename` is the output file name as a command line argument. The report is written to the file name. If no file name is specified, then the output is written to STDOUT.
- `-c CSC Root` is the CSC root for the license check and courier counting.
- `-a` generates the audit report.

hcilicset

This command enters a machine's license key and creates a `license.dat` file based on the feature names and keys. After the information is entered, `hcilicset` calls `hcilictest` to verify that the license is valid.

```
hcilicset license key
```

hcilicstatus

This command checks clients' licenses individually. Each file is loaded and the license key is selected.

```
hcilicstatus [{-p | {-m | -c | -a}
module }]
```

- No args prints a summary of the product license status.
- `-p` checks if the general product is licensed.
- `-m module` checks if the Message module is licensed.
- `-c module` checks if the Communication Protocol module is licensed.
- `-a module` checks if the Add-On module is licensed.

Consult your Account Manager if the return value states that you do not have a license installed.

The command output also shows the running protocol/connection threads. For example:

```
> hcilicstatus

Thread conn_1 in site helloworld is running.
Thread conn_2 in site helloworld is running.
Thread conn_11 in site helloworld is running.
Thread conn_3 in site helloworld is running.
Thread conn_4 in site helloworld is running.
Thread conn_111 in site helloworld is running.
Thread conn_21 in site helloworld is running.
Thread conn_12 in site helloworld is running.
Thread conn_1 in site t-ar10531d is running.
Thread out in site t-ar10531s is running.
Thread in in site t-ar10531s is running.
Thread receivedata_tcpip in site t-ar6630 is running.
Thread receivedata_file in site t-ar6630 is running.
Thread senddata_filesetlocal in site t-ar6630 is running.
Thread senddata_tcpip in site t-ar6630 is running.
Thread conn_21 in site testaa is running.
Thread conn_11 in site testaa is running.
Thread conn_1 in site testab is running.
Currently there are 18 threads running under HCIR00T(C:\infor\cis6.2P\integrator).

Package Name: Infor Cloverleaf(R) Integration Services Enterprise Plus

Interfaces
cl-intfc-master: yes
cl-intfc-thread-0: no

Message Models
cl-mm-edifact: no
cl-mm-frl: no
cl-mm-hl7: no
cl-mm-hprim: no
...
```

```
...
```

hcilicest

This command queries the machine's license key. The output is "OK" if the license key is valid.

```
hcilicest package
```

Example output:

```
cl-pkg-ent-plus: OK
```

hcilmclear

This command cleans up after a crashed engine that has not properly disconnected from the Lock Manager.

```
hcilmclear {-s | -p process -m | { -A | -a tablename }} [-v]
```

- `-s` prints a status report for all database users of the Lock Manager.
- `-p process` specifies the name of the engine process to clean up.
- `-m` cleans up the database user for `hcimonitor`.
- `-A` cleans up all database users for `hcimsgarchive`.
- `-a tablename` cleans up the database user that is specified by *tablename* for `hcimsgarchive`.
- `-v` specifies verbose mode.

This command clears up database problems, such as `db_vista -921`, which indicates the user is already in the database from a failure to previously log out. This can happen when a process panics or is otherwise stopped during running.

This command also eliminates the requirement to shut down the entire site to cycle the Lock Manager. This happened when a process crashed or did not shut down properly.

In most cases, this command should run automatically, but the command can also be run manually.

This is valid for Windows and UNIX users.

hcils

Valid `hci` list commands are:

- `hcialertls`.
See [hcialertls](#).
- `hcieols`.
See [hcieols](#).

- `hcifrlls`.
See [hcifrlls](#).
- `hcihrlls`.
See [hcihrlls](#).
See [hcitblls](#).
- `hcitblls`.
- `hcivrlls`.
See [hcivrlls](#).
- `hcixltls`.
See [hcixltls](#).

Usage is listed under each command.

hcimonitor

This command enables the Monitor Daemon.

```
hcimonitor [-de eoc-pattern] [-dd eoc-pattern] [-cl cfg-list]
            [-nl] [-D] [-S sitename]
```

- `-de eoc-pattern` enables an engine output configuration pattern.
- `-dd eoc-pattern` disables an engine output configuration pattern.
- `-cl cfg-list` specifies a list of alert configuration files to process. Use a space to separate multiple file names in the list.
If no file name is specified, then `default.alert` is processed, if it exists.
- `-nl` enables non-daemon mode. This indicates no log file. In this case, the Monitor Daemon runs in the foreground.
This option is used only for debugging.
- `-D` enables debugging information.
- `-S sitename` is the site name.

hcimsgarchive

This command works on the data from the message archive database cache.

```
hcimsgarchive {-l | -L | -w | -d | -v | -c-c | -e} [-V] [-c templatename]
               [-I templatename] [-p proc] [-t tablename] [-s threadname]
```

- `-l` lists the messages waiting to be updated to the external database. This outputs the list and the number of messages that are waiting to be written to the external database.
- `-L` lists messages with Ultra Long form. Overrides `-l`.

- `-w` writes the messages to the external database. This outputs the list and the number of messages that are written to the external database.
- `-d` deletes the messages without writing to the external database. This outputs the list and the number of message that are deleted.
- `-v` verifies the connection to the external database. This tests the external database configuration. If the table does not exist in the external database, then the tool attempts to create the database using the table creation template. A row with placeholder data is inserted. The MID for this row is "0.0.0".
- `-e` verifies whether a table exists in the external database.

For example:

```
hcimsgarchive -e -t tableName
```

- `-c` cycles the log file.
- `-v` is verbose mode.
- `-C templatename` specifies the template to create the database table.
- `-I templatename` specifies the template for inserting data to the external database.
- `-p proc` is the Tcl procedure to modify the message data. If this is not specified, then the message is processed as usual.
- `-t tablename` is the external database table name. This can be combined with `-v`, `-w`, or `-e`. If not specified, then this processes all tables or messages.
- `-s threadname` is the protocol thread name. This can be combined with `-l`, `-L`, `-d`, `-w`, or `-p`. If not specified, then this processes all tables or messages.

The `hcimsgarchive` command runs the Tcl procedure before writing the message to the external database. After the procedure is run, `hcimsgarchive` inserts the values into the external database that are based on the returned values.

The procedure is valid when writing the message to the external database. For the other actions, such as `list` or `delete`, it is invalid.

marp UPoC interface

The input argument is a list with one element:

```
args := insertion_template_list
```

This procedure returns a value in the same form. The return value can contain multiple *insertion_template_list* keyed lists where each one is surrounded by curly braces.

```
retval := {insertion_template_list}[ {insertion_template_list}]*
```

insertion_template_list is a keyed list describing the default template.

These keys are expected to be in the keyed list:

- **TEMPLATE:**
 - The insertion template name
 - Required: Yes

- Default Value: N/A
- Input Value: In the inbound args, `TEMPLATE` is populated with the *templatename* from the `-t` argument. If `-t` is not used, then it is populated with the default template.
- `*LIST` keys:
These keys are used to define the names, types, and values that are used to substitute fields in the template.
For example, if *MID* is found in the template, `hcmsgarchive` uses the type and value corresponding to *MID* in the lists to insert the value.
- `NAMELIST`:
 - Required: Yes
 - Default Value: N/A
 - Input Value: The names available by default when no procedure is specified.
- `TYPELIST`:
 - Required: Yes
 - Default Value: N/A
 - Input Value: The type list corresponding to `NAMELIST`.
- `VALUelist`:
 - Required: Yes
 - Default Value: N/A
 - Input Value: The value list corresponding to `NAMELIST`.
- `FAILACTION`:
 - Error handling for this inserting.
 - Required: No
 - Default Value: `rollback`
 - Input Value: Does not exist for input.
This key describes how `hcmsgarchive` should behave if insert fails.

```
{FAILACTION action}
```

The value of `FAILACTION` should be one of:

- `skip`
If the insert fails and `FAILACTION` is `skip`, `hcmsgarchive` moves on to the next insert. For example, if there are three tables with data to be inserted and the second insertion fails, the second table insertion is skipped. The first and third insertions are committed.
- `rollback`
If the insert fails and `FAILACTION` is `rollback`, `hcmsgarchive` stops insertion and rolls back all insertions that are performed up to this point. For example, if there are three tables with data to be inserted, and the second insertion fails, the first insertion is rolled back. This is the default if no `FAILACTION` is specified.
- `template_name`
If the insert fails and `FAILACTION` is not `skip` or `rollback`, `hcmsgarchive` attempts to redo the insert with the template name that is provided. If necessary, then use an update template as the alternate if an insertion fails.

marp template

This is the template for procedure type `marp` that is called by the `hcimsgarchive` command is:

```
#####
# Name:          __MODULE_NAME__
# Purpose:       This proc called by hcimsgarchive command to modify the
#               message data to be saved to the external Database.
# UPoC type:     marp
# Args:          keyed list containing these keys:
#               TEMPLATE      The template specified for inserting.
#               NAMELIST      The name list for inserting.
#               TYPELIST      The type list used in external database.
#               VALUelist     The values list for inserting
# Returns:       list of keyed lists containing these keys:
#               TEMPLATE      The template specified for inserting.
#               NAMELIST      The name list for inserting.
#               TYPELIST      The type list used in external database.
#               VALUelist     The values list for inserting
#               FAILACTION    Error handling for insertion failure.
# Notes:         <put your notes here>
#
# History:       <date> <name> <comments>
#
proc __MODULE_NAME__ { args } {
    keylget args TEMPLATE template      ;# Fetch template

    return {$args}
}
```

This template, if left unchanged, except for `__MODULE_NAME__`, uses the same values passed from the input. The procedure does not change the insertion behavior.

This template file is located in `$HCIRoot/tclprocs` named `marp_template`.

Specifying a UPoC

When using `hcimsgarchive` and specifying a UPoC, the UPoC returns three Tcl list, `NAMELIST`, `TYPELIST`, and `VALUelist`, to the caller.

To prevent duplicates, messages in EBCDIC, messages with headers, and so on, you can manipulate the data before it is stored to the external database. For example, you can parse an HL7 message into its component parts and store those into separate fields.

Data that is inserted into the returned lists must not contain spaces or be enclosed in brackets. This makes it a distinct single list element.

If a space is added in the message content by a procedure, then this message cannot be archived completely to the external database. Tcl uses a space as the delimiter in a list, so add "{}" for any fields that include a space.

The `hcimsgarchive` command prompts a warning message if the fields `VERSION`, `SEPCHARS`, `RECORDFORMAT`, `USERDATA`, `DRIVERCONTROL`, and `DATA` contain a space in the returned list.

Note: Check whether the value for `DATA` contains whitespace and add an extra set of {} around the value if necessary.

To help you diagnose the issue, this warning is given for fields that might commonly contain spaces.

This can happen on any field that might contain a space. The space is not necessarily added by the user during the procedure; it might be part of the incoming data.

For example, a space is added to message data by changing `VALUELIST` in the procedure. Then, `hcimsgarchive -w -p upoc` is run. The data then truncates after the first space. For example, "test message 1" is truncated to "test".

To get around this, you would add "{}" for the fields which includes the space:

```
set name1 "[list TABLENAME MID VERSION SITE PROCESS SRCTHREAD CONTEXT TIMESTORED TIMEARCHIVED
DATA]"
set value1 "[lindex $vl $iTableName] [lindex $vl $iMID] {[lindex $vl $iVer]} [lindex $vl $iSite]
[lindex $vl $iPro]
[lindex $vl $iSRCThread] [lindex $vl $iCon] [lindex $vl $iTimeS] [lindex $vl $iTimeA] {[lindex
$vl $iData]}"
```

msgarchive UPoC special characters

Special characters, including ', \, {, and } are escaped in the msgarchive UPoC. The message content is kept with no change and archived in the external database.

Sample return values

The list content is used to populate the template fields and set the column data types.

Empty list

If the return value is an empty list, then `hcimsgarchive` skips the record insertion and removes it from the archive database. This is equivalent to using the `-d` option.

Single keyed list

This is a sample return value that returns a single insert using the `msginsert` template with MID and Message Data for the value list. If the insert fails, then the archive is retried with the alternate template `msgupdate`.

```
{
  {TEMPLATE msginsert}
  {NAMELIST {MID MSGDATA}}
  {TYPELIST {SQL_CHAR SQL_CHAR}}
  {VALUELIST {0.0.152 {MSH|...}}}
  {FAILACTION msgupdate}
}
```

Multiple keyed lists

This is a sample return value that uses a complex mapping between parsed message data and multiple tables.

```
{
  {TEMPLATE msginsert}
  {NAMELIST {TABLENAME MID MSGDATA}}
  {TYPELIST {SQL_CHAR SQL_CHAR SQL_CHAR}}
  {VALUELIST {MessageData 0.0.152 {xxxxxxxxxxxxxxxxxxxxxxxxxxxx}}
}
{
  {TEMPLATE patientinsert}
  {NAMELIST {TABLENAME PATIENTID ...}}
  {TYPELIST {SQL_CHAR SQL_CHAR ...}}
  {VALUELIST {PatientInfo 321 ...}}
}
```

If there are multiple tables, then the inserts are run in the list order.

Log files

This program is designed to run in the background in the update mode. If there are problems, then the errors and warnings are written to a log file located in `$HCISITEDIR/exec/hcimsgarchive`.

- If a table name is specified, then the log messages, warnings, and errors are written to `tablename.log` and `tablename.err`.
- If no table name is specified, then the log messages, warnings, and errors are written to `hcimsgarchive.log` and `hcimsgarchive.err`.

These log files are cycled automatically when the size reaches the limit defined in `siteInfo`.

The log files can also be cycled using `hcimsgarchive` with the `-c` option.

Concurrent archiving

With `hcimsgarchive`, you can do concurrent archiving to different tables. There cannot be two simultaneous archive processes to the same table.

Message archiving checks if a particular table is being archived. If it is, then it skips over that table and continues to the next table, if applicable.

External SQL database templates

There is an `archiving` folder under the root and site level. This folder contains the SQL template scripts for table creation and insertion.

These templates are compatible with Oracle 11. Other external databases may require different data types.

The scripts that are described here are installed at the root level. To customize the table creation or data insert, copy these templates to the site level to customize for a site. You can also copy them to the master site level to customize for a group of sites.

- If the template file is not found in `$HCISITEDIR/archiving`, then it searches the master site.
- If it is not found there, then it searches `$HCIR00T/archiving`.
- If the template file is not found in any of these locations, then an error is thrown and the archiving does not complete.

Template fields

In these template files, there are strings that are replaced when used by the `hcimsgarchive` tool as SQL statements. These fields are identified with a matching pair of `<` and `>` characters. For example, `<TABLENAME>` is replaced with the table name.

If the field is not supported, then the text is not substituted.

If the resulting text is not a valid SQL statement, then an error results when `hcimsgarchive` attempts to run it.

All fields available from the message archive embedded database are supported.

Table creation template

If the data source user has table creation permissions, then the table is created using this template the first time `hcimsgarchive` connects. If the data source user does not have table creation permissions, then the table must be created independently. This must be created by someone who has permission, for example, a database administrator.

This is a portion of the template for the table creation SQL that is used to insert data. `<TABLENAME>` is replaced with the table name. The items that are surrounded by `<` and `>` characters are replaced with the equivalent bind parameters.

```
CREATE TABLE <TABLENAME> (
  MID          VARCHAR(16)    NOT NULL,
  SOURCEMID    VARCHAR(16)    NULL,
  TYPE         VARCHAR(16)    NULL,
  PRIORITY     VARCHAR(16)    NULL,
  HOSTID       VARCHAR(128)   NOT NULL,
  VERSION      VARCHAR(16)    NOT NULL,
  SITE         VARCHAR(64)    NOT NULL,
  . . . . .
```

This template is stored as `create_template.sql` in the archiving folder.

Data insertion template

This is a portion of the template for the table insertion SQL that is used to insert data.

```
INSERT INTO <TABLENAME> (
  MID          ,
  SOURCEMID    ,
  TYPE         ,
  PRIORITY     ,
  VERSION      ,
  SITE         ,
```

```

HOSTNAME      ,
HOSTID        ,
PROCESS       ,
THREADNAME    ,
SRCTHREAD     ,
DESTTHREAD    ,
. . . . .

```

This template is stored as `insert_template.sql` in the `archiving` folder.

SiteInfo

This section is found in `SiteInfo` to configure the message archiving data source.

These are used to create an JDBC connection to the data source.

The password is encoded in base64.

Note: The `archive_logcycle` key in `SiteInfo` is shared by message archive and tracing.

```

#####
Cloverleaf Message Archiving Configuration
archivedbconnection - The database connection name
archivedbconnection=
#####
archive_logcycle - The size (in kb) the archive_log should be cycled
and this key is also shared with Message Tracing
logcycle as threshold.
0 : maximum size is unlimited, 0 is
also the value by default. It means unlimited.
x (x>0): the maximum size is set.
archive_logcycle=0

```

hcmsgtrace

This command enables tracing of messages as far as information is recorded in the tracing database.

```
hcmsgtrace {-l|-L|-w|-d|-v}
```

- `-l` lists messages waiting to be written to external database.
- `-L` list messages with Long format.
- `-w` writes messages to the external database.
- `-d` deletes messages without writing to the external database.
- `-v` verifies the connection to the external database.

hcimsiutil

This command displays the information usually found in a thread's status from Network Monitor. There is also an option to reset only the error count.

```
hcimsiutil [-D] [-dh] [-dt] [-da] [-dd thread] [-pd] [-f] [-R] [-rs] [-ur] [-ut thread]
[-E] [-ep proc] [-et thread] [-X] [-xt thread] [-xp proc] [-Z]
[-zt thread] [-zp proc]
```

- `-D` enables debug output.
- `-dh` dumps region header.
- `-dt` dumps region Table of Contents.
- `-da` dumps all. This includes the region header and table of contents.
- `-dd thread` dumps data section for thread.
- `-E` clears all of the thread's error count.
- `-ep proc` clears all of the thread's error count within a specified process.
- `-et thread` clears the error count of a specified thread.
- `-f` removes/resets thread data. This is used with `-E/e`, `-R/-rs`, `-X/x` or `-Z/z`.
- `-pd proc` marks all threads in process as dead.
- `-R` removes the region.
- `-rs` removes the semaphores only.
- `-ur` unlocks the region control semaphore.
- `-ut thread` unlocks thread section semaphore.
- `-x` keeps the times but resets all other thread section data.
- `-xt thread` keeps the times but resets other thread section data.
- `-xp proc` keeps the times but resets all other thread section data of one process.
- `-Z` resets all thread section data.
- `-zt thread` resets thread section data.
- `-zp proc` resets all thread section data of one process.

Safe shared memory reset: `-X/-xt`

To avoid undesirable alerts that come from resetting shared memory time values, use these options that leave these values alone.

- `-x` is the same as the existing `-z` option, except no time values in the shared memory are reset.
- `-xt thread name` corresponds to the exiting `-zt` option, but all time values are retained.

These values are unaffected by the safe shared memory reset:

- Sample Taken
- Sample Added
- Start Time
- Stop Time
- Proto Last Rd
- Proto Last Wt
- Proto Err Time

Values for Proto Status are:

- INITIALIZING
Value: 0
- OPENING
Value: 1
- UP
Value: 2
- DOWN
Value: 3
- CLOSING
Value: 4
- ERROR
Value: 5
- INEOF
Value: 6
- TIMER
Value: 7
- STATUS
Value: 8

For Proto Flags, only the "hold" flag is set. When the flag is set (1), the outbound protocol holds messages.

hcimvf

Note: The related script `hcimvf` is located under `$hciroot/bin`.

`hcimvf` (Migration Validation Framework) is a regression testing tool that automatically processes SMAT file/Database and gets route test outputs. This tool can also compare two test outputs.

```
hcimvf -testid value -smatfile value [-smatdbkey value]  
[-obsmatdbkey value] [-smatencoding value] -thread value [-verbose] [-output value]
```

- `-testid value` specifies the test ID.
- `-smatfile value` specifies full-path of SMAT file. Only one file is supported. Database SMAT must end with `.smatdb`.
- `[-smatdbkey value]` specifies the SMAT Database key. This is treated as unencrypted if not specified or empty.
- `[-obsmatdbkey value]` specifies the outbound encryption key. This is treated as unencrypted if not specified or empty.
- `[-smatencoding value]` specifies the encoding of message content. The default is UTF-8.
- `-thread value` specifies the thread name to do the route test.
- `[-verbose]` specifies logging in verbose format.

- [-output value] specifies the full path for the output file.

The default is \$HCISITEDIR/exec/hcimvf/testid/output/testid.sqlite.

Or

```
hcimvf -testid value -old value [-oldkey value]
-new value [-newkey value] [-verbose] [-output value]
```

- -testid value specifies the test ID.
- -old value specifies the full path of the old test output file.
- [-oldkey value] specifies the encryption key for the old test output file.
- -new value specifies the full path of the old test output file.
- [-newkey value] specifies the encryption key for the new test output file.
- [-verbose] specifies logging in verbose format.
- [-output value] specifies the full path for the output file.

The default is \$HCISITEDIR/exec/hcimvf/testid/output/output.rpt.

Notes

hcimvf works when copying to the 19.1 and 6.2 versions. To do this:

- 1 Copy \$HCIRoot/bin/hcimvf* to the lower version under the same directory.
- 2 Modify the first line of the script with the current root name.
- 3 After invoking hcimvf, testid dir is created under \$HCISITEDIR/exec/. The log, status, and output are located in this directory.

Command limitation: When two test outputs are compared and are different, the source message cannot be located; only the batch that contains the source message can be located.

Limitation example:

- The routed message identifier is: \$ibthread-\$batchIndex-\$obthread-\$msgIndex.
- batchIndex is used for performance considerations.
- The batch number is 1000.
- hcimvf invokes hciroutetest, but cannot back trace if batch messages are used as input.
- Example:

- 1 Process SMAT Database and get the route test output. Example:

```
hcimvf -testid test_route -smatfile c:/cloverleaf/cis20.1/integrator/t-hcimvf/exec/pro
cesses/
multiple_route/ib.smatdb -smatdbkey t-hcimvf -obsmatdbkey t-hcimvf -thread src
```

- 2 Compare two test outputs:

```
hcimvf -testid test_diff -old c:/cloverleaf/cis20.1/integrator/t-hcimvf/ex
ec/hcimvf/cmp1/output/
cmp1.sqlite -oldkey t-hcimvf -new c:/cloverleaf/cis20.1/integrator/t-hcimvf/ex
ec/hcimvf/cmp2/output/
cmp2.sqlite -newkey t-hcimvf
```

Note: File SMAT does not include .msg or .idx.

hcincdpdptest

This tests NCPDP Telecom configurations against actual data.

```
hcincdpdptest [-a] [-e ] [-d n] [-i] [-f format] [-V version]
[-v variant] [-t test] [-n] [-F fieldname] file
[-w search expression] [-p sepChars] [-c drivercontrol]
```

- `-a` processes all records in the file.
- `-e encoding` is the encoding name for the data file.
- `-d n` shows data at detail level *n*. The detail level range is 0-4, where "0" is raw, unparsed data and "4" is the most detailed.
- `-i` displays field addresses using index notation.
- `-f format` specifies the file format:
 - `-f len10` specifies 10-byte length-encoded.
 - `-f nl` specifies newline terminated. This is the default.
 - `-f eof` specifies end-of-file terminated.
- `-V version` specifies the NCPDP Telecom version.
- `-v variant` specifies the NCPDP Telecom variant.
- `-t test` specifies the test to run:
 - `-t numbers` tests *Nn* number conversions.
 - `-t parse` test parses a message. This is the default.
- `-c drivercontrol` is used to set the content to message metadata DRIVERCONTROL.
- `-n` shows the field names.
- `-F fieldname` specifies the field name only.
- `-w search expression` specifies the search string or regular expression for which to search.
- `-p sepChars`.
- *file* specifies the data file with which to test. are the separator characters used for parsing or encoding. This needs to be in a keyed list format.

hcincdpdpscripttest

This tests NCPDP Script configurations against actual data.

```
hcincdpdpscripttest [-a] [-d n] [-e encoding] [-i] [-f format]
[-V version] [-v variant] [-t test] [-c drivercontrol]
[-n] [-F fieldname] [-w search expression] [-p sepChars] file
```

- `-a` processes all records in the file.
- `-e encoding` is the encoding name for the data file.
- `-d n` shows data at detail level *n*. The detail level range is 0-4, where "0" is raw, unparsed data and "4" is the most detailed.
- `-i` displays field addresses using index notation.
- `-f format` specifies the file format:
 - `-f len10` specifies 10-byte length-encoded.
 - `-f nl` specifies newline terminated. This is the default.

- `-f eof` specifies end-of-file terminated.
- `-V version` specifies the NCPDP Script version.
- `-v variant` specifies the NCPDP Script variant.
- `-t test` specifies the test to run:
 - `-t numbers` tests Nn number conversions.
 - `-t parse` test parses a message. This is the default.
- `-c drivercontrol` is used to set the content to message metadata DRIVERCONTROL.
- `-n` shows the field names.
- `-F fieldname` specifies the field name only.
- `-w search expression` specifies the search string or regular expression for which to search.
- `-p sepChars` are the separator characters used for parsing or encoding. This needs to be in a keyed list format.
- `file` specifies the data file with which to test.

hcincdpfabtest

Defines NCPDP Formulary and Benefit configurations against actual data.

```
hcincdpfabtest [-a] [-d n] [-e encoding] [-i] [-f format]
[-V ] [-v variant] [-t test] [-c drivercontrol] [-n] [-F fieldname]
[-w search expression] [-p sepChars] file
```

- `-a` processes all records in the file.
- `-e encoding` is the encoding name for the data file.
- `-d n` shows data at detail level *n*. The detail level range is 0-4, where "0" is raw, unparsed data and "4" is the most detailed.
- `-i` displays field addresses using index notation.
- `-f format` specifies the file format:
 - `-f len10` specifies 10-byte length-encoded.
 - `-f nl` specifies newline terminated. This is the default.
 - `-f eof` specifies end-of-file terminated.
- `-V version` specifies the NCPDP Formulary and Benefit version.
- `v variant` specifies the NCPDP Formulary and Benefit variant.
- `-t test` specifies the test to run:
 - `-t numbers` tests Nn number conversions.
 - `-t parse` test parses a message. This is the default.
- `-c drivercontrol` is used to set the content to message metadata DRIVERCONTROL.
- `-n` shows the field names.
- `-F fieldname` specifies the field name only.
- `-w search expression` specifies the search string or regular expression for which to search.
- `-p sepChars` are the separator characters used for parsing or encoding. This needs to be in a keyed list format.
- `file` specifies the data file with which to test.

hcinetcheck [Netconfig]

This checks the Network Configurator file for configuration errors.

When message archiving is enabled for a thread, the NetConfig validation searches for a matching alert in all alert files. If one is not found, then a warning is issued:

Message archiving has been enabled for thread *threadname* to table *table* but no matching alert definition to schedule the archiving is found.

hcinetconvert

This upgrades from one major release to another when the format for Network Configurator has changed.

hcinetdiff

This compares the current site against another.

```
hcinetdiff baseNetConfigpath
```

baseNetConfigpath specifies the full path to the name of the Network Configurator file against which to compare the current site.

hcidpc

This compiles PDL scripts.

```
hcidpc [-Wno- warn_tag] [-O dir] [-D debug_tag]
[-Dno- debug_tag] files
```

- `-Wno-` disables the specified warnings.
- `warn_tag` specifies the warn tag:
 - `eof` is an unexpected end-of-line (lexical).
 - `eof` is an unexpected end-of-file (lexical).
 - `empty` is a character class that is empty or a zero-length fixed array.
- `-O dir` sets the output directory.
- `-D` enables the specified debugging output.
- `-Dno-` disables the specified debugging output.
- `debug_tag` specifies the debug tag:
 - `lex` specifies lexical information. This is not enabled with `all`.
 - `parse` specifies parsing information.
 - `top` specifies second pass interpretation information.
 - `phrase` specifies phrase computation information.
 - `codegen` specifies code generation information.

- `dfa` specifies data fragment information.
- `device` specifies device information. For example, `async`, `tcp-server`.
- `all` enables all tags, except `lex`.

hciprocedit

This command starts an editing session to create Tcl scripts for use within the engine. It starts an editing session with a template of the UPoC type that is specified by the `-create` argument. The default editor is `vi` in UNIX; Notepad in Windows. To override the default editor, set the `EDITOR` variable to the editor of choice.

You can use one of these commands:

```
hciprocedit [-xterm | -gui] [-create type] proc
```

```
hciprocedit [-xterm] [-create type] file
```

- `-xterm` starts an Xterm to run the editor. Useful when calling this script from an external GUI.
- `-gui` is a synonym for `-xterm`.
- `-create type` specifies the type of the new script file, where `type` is one of:
 - `routetest` specifies the end procedure.
 - `tps` specifies the TPS module.
 - `tpstest` specifies the `hcitptest` end procedure.
 - `trxid` specifies the transaction ID determination procedure.
 - `xltp` specifies the translation procedure that is called from within an `xlate` code fragment.
- `proc` specifies the name of the selected procedure.
- `file` specifies the base name of the file to edit. The file name must include the `.tcl` extension.

hciprocstatus

This command checks the status of an engine process.

```
hciprocstatus [-p process] [-i] [-e]
```

- `-p process` specifies all connections in `process`.
- `-i` displays the entire process name without truncation.
- `-e` exits non-zero if the process is running.

The default output is a summary of all processes.

hcireglist

This command is used to look up the Cloverleaf RMI objects that are bound to a remote object registry on a specified `host` and `port`.

- If a host is not specified, then "local host" is used.
- If port is not specified, then "1099" is used.

Usage:

```
hcireglist [ip or host name] [port]
```

Example usage and output:

```
C:\Users\cch>hcireglist usspabcd 13024

Registry=RegistryImpl_Stub[UnicastRef [liveRef: [endpoint:[usspabcd:13024](remote),objID:[0:0:0,0]]]]

AUTH_CloverleafGMSServer_1.0
AUTH_CloverleafServer_1.0
AUTH_MonitorDSERVER_1.0
AUTH_XltDebugServer_1.0

AUTH_CloverleafGMSServer_1.0
Proxy[GMSServer,RemoteObjectInvocationHandler[UnicastRef2 [liveRef:
[endpoint:[10.46.40.103:53282,com.hie.util.tls.RMITLSClientSocketFactory@4534b60d]
(remote),objID:[37e29ccd:174a07aca5e:-7ffb, -8450094591385734997]]]]]

AUTH_CloverleafServer_1.0
Proxy[CloverleafServer,RemoteObjectInvocationHandler[UnicastRef2 [liveRef:
[endpoint:[10.46.40.103:53282,com.hie.util.tls.RMITLSClientSocketFactory@4534b60d]
(remote),objID:[37e29ccd:174a07aca5e:-7ffb, 1358943836246994187]]]]]

AUTH_MonitorDSERVER_1.0
Proxy[RMITMonitorDSERVER,RemoteObjectInvocationHandler[UnicastRef2 [liveRef:
[endpoint:[10.46.40.103:53282,com.hie.util.tls.RMITLSClientSocketFactory@4534b60d]
(remote),objID:[37e29ccd:174a07aca5e:-7ffd, -5770908384333419398]]]]]

AUTH_XltDebugServer_1.0
Proxy[RMITXltDebugServer,RemoteObjectInvocationHandler[UnicastRef2 [liveRef:
[endpoint:[10.46.40.103:53282,com.hie.util.tls.RMITLSClientSocketFactory@4534b60d]
(remote),objID:[37e29ccd:174a07aca5e:-7ffb, 5913916851892193038]]]]]
```

hcireloadobjects

This command does not natively reload objects until one of these actions is completed:

- Option `-r` is specified in the invocatio.
- `hcireloadobjects=1` is set in `rootInfo`.

By default, `hcireloadobjects` reloads all running processes under the current site.

Usage:

```
hcireloadobjects [-s site] [-a] [-r]
```

- `-s site` reloads objects in all running processes under `site`.
- `-a` reloads objects in all running processes under the current root.
- `-r` enables `hcireloadobjects`.

The `hcireloadobjects` variable in `rootInfo` can also enable `hcireloadobjects` when this is set to 1.

`hcireloadobjects` is disabled when set to 0.

An error message is given when the function is disabled:

The “reload all objects” function is currently disabled. Please contact your system administrator to enable it if required.

hcirootcopy

This command copies sites from old roots to the currently used root. This includes the version folder. This is for version control.

When a user adds a configuration into version control, a version folder is created in the current site to store the configuration.

Note: `hcirootcopy` copies all folders except `exec`.

```
hcirootcopy [-f] [-n] [-v] [-N] [-s sitelist] [-l listfile]
[-p path] [-e encoding] [-u username] [-w password]
[-E SMATDBFlag,InternalDBFlag,ErrDBFlag] [-D type] [-r] [-c] sourceRoot
```

- `-f` specifies no query.
- `-n` specifies test query.
- `-v` specifies verbose mode.
- `-N` specifies to not copy the database when copying from root.
- `-s sitelist` copies a single site or colon-separated list of sites. For example, `site1` or `site1:site2:site3`
- `-l listfile` copies the sites that are listed in `listfile`.

`listfile` should be newline formatted. For example:

```
site1
site2
site3
```

- `-p path` is the actual path where `sitelist` is to be saved.

A `$HCIR00T/site` symbolic link under the current root is created as a blank placeholder file to the actual path of `site path/site` after site creation.

This option is supported only on UNIX.

- `-e encoding` is the encoding of the source files. If this option is omitted, then the default value is `utf-8`.
- `-u username` is your organization's end-user user name.
- `-w password` is your organization's end-user password.
- `-E SMATDBFlag,InternalDBFlag,ErrDBFlag`

These specify whether the SMAT, internal, or error database is encrypted.

`SMATDBFlag` is used for the SMAT database.

`InternalDBFlag` is used for the Raima database.

`ErrDBFlag` is for error database:

- `1` indicates encrypted.
- `0` indicates unencrypted.

- `-i` indicates no change. This is the default. If no flag is specified, then it keeps the same configuration as the previous site.
For example, `-E -i,1,1` indicates that SMAT encryption has no change after migration, and the internal/error databases are encrypted after migration.
- `-D type` is the driver type that is used to read/write SMAT. Acceptable values are `file` or `sqlite`.
If `-D type` is `sqlite`, then the `sqlite` database retains the same behavior as the internal and error databases.
- `-r` copies the `server/server.ini` file.
Note: This option ignores the security server-related keys.
- `-c` updates the encryption method.
- `sourceRoot` specifies the source root directory.

Notes

- Options `-s` and `-l` cannot be used simultaneously.
- Option `-p` is ignored if the site already exists in the current root.
- To update the encryption method:
 - 1 Provide the user name and password.
 - 2 Upgrade to basic security mode.
 - 3 Enable CLAPI.
 - 4 Start the host server.

Conversion commands

These conversion commands are in the source code of `hcirootcopy`:

- `hcihl7convert`
Converts HL7 configurations to the current version.
This renames `$siteName/hl7_v2.1` to `$siteName/formats/hl7/2.1`. `hl7_v2.1` should be in an old version.
- `hcixltconvert`
Converts `xlt` files.
If migrating from pre-5.8 to 5.8 or above, then `hcirootcopy` runs `hcixltconvert` to convert `xlt` files.

These commands are not run in `hcirootcopy`. You must run these commands manually by themselves, if necessary.

- `hcitpsconvert`
If migrating from pre-3.8 version to 3.8 or above, then you must run `hcitpsconvert` manually to convert Tcl procs.
- `hcigdbmconvert`
If migrating from pre-3.9 version to 3.9 or above, then you must run `hcigdbmconvert` manually to convert null-terminated databases to non-terminated.
- `hcix12convert`
If migrating from 3.4 version to 3.5 or above, then you must run `hcix12convert` manually to convert all format translation files from an earlier system version to a later one.

- `hciedifactconvert`

If migrating from 3.4 or pre-version version to 3.5 or above, then you must run `hciedifactconvert` manually to convert the UN/EDIFACT configuration files to the current version.

- `hcismatconvert`

If migrating from 3.4 or pre-version version to 3.5 or above, then you must run `hcismatconvert` manually to convert SMAT files between formats: old to new, or new to old.

hciroutetest

This command tests interfaces. `hciroutetest` simulates the route from a particular source thread. In this simulation, all TPS modules and translations that are used by the source thread are enacted.

`hciroutetest` runs any inbound TPS for the source thread before it passes the messages to `route` and `translate`. This is necessary because the TPS often sets up metadata that is required for further processing.

For `hciroutetest` to run `translate` procs in startup mode, select the **Run Translation Procs in Start Mode** checkbox on the **Process Configuration** dialog box. This is found at **Process > Configure** in Network Configurator.

```
hciroutetest [-a] [-x encoding] [-d] [-t] [-e "proc args"]
[-p "proc {args}"] [-f format] [-c drivercontrol]
[-w search_expression] [-L] [-i] [-n] [-r thread] [-s savebase]
thread file
```

- `-a` processes all records in the file.
- `-x encoding` encodes from *encoding* to UTF-8.
- `-d` processes as DATA messages.
- `-t` runs the outbound TPS. When this is defined, only outbound TPS in the same site is run; all inter-site threads are ignored.
 - If `-t` is specified, then there must be a defined outbound TPS; otherwise, a warning results. Only the TPS that is in the same site is run; no inter-site.
 - All outbound TPS that are defined in the route path are run once with start mode. Be cautious with global variables.
 - If both `-t` and `-e` are defined, then the outbound TPS are run before `-e`, so that output messages become input messages to the procedure.
 - TPS uses the same disposition policy as translation post queue. The messages for KILL, OVER, and ERROR dispositions are discarded.
 - OB TPS and **Send to Proc** use different Tcl interpreters. OB TPS uses the same Tcl interpreter as IB TPS.
- `-e "proc args"` specifies end processing configuration.
- `-p "proc {args}"` specifies error processing configuration. The specified error TPS is used when any message fails to do `xlate/route`.

If `-p` is given, then `hciroutetest` uses the given error TPS to handle any failed messages.

If `-p` is not given, then `hciroutetest` uses the error TPS defined in NetConfig.

This option catches any message which fails in `xlate/route`; only error information is shown in `stderr`.

- `-f format` specifies the input file format.
 - `-f len10` specifies 10-byte length-encoded.

- `-f nl` specifies newline terminated. This is the default.
- `-f eof` specifies end-of-file terminated.
- `-c drivercontrol` is used to set the content to message metadata DRIVERCONTROL. You can input any content to set to *drivercontrol* using this option.
- For Route test on DICOM messages, `-c drivercontrol` is required, where *drivercontrol* is {AbstractSyntax***} {TransferSyntax***}. This is required to get the TRX ID. See [DICOM abstract syntax values](#) and [DICOM transfer syntaxvalues](#).

Example:

```
hciroutetest -a -x UTF-8 -f eof -c "AbstractSyntax 1.2.840.10008.5.1.4.1.1.4}
{TransferSyntax 1.2.840.10008.1.2.4.91}" CL_SCP c_store.dat
```

- `-w search_expression` specifies the search string or wildcard for which to search. Only the lines of output matching the search expression are printed out.
- `-L` specifies Tcl handle leak detection.
- `-i` shows the input source message ID in stdout.
This shows information about the source message so that when `-a` is used, you can know the relationship between the output message and input message.
- `-n` shows intermediate branched and chained messages.
- `-r thread` processes as REPLY to thread.
- `-s savebase` specifies the Save File base name.
- *thread* specifies the source thread to simulate.
- *file* specifies the data file with which to test.

Note: This information displays also on the Testing Tool in Preview Command to Issue as the test is built.

Global variable support

Global variable support is available for the `hciroutetest` command line tool. The global variable indicator `$$` has a conflict with the shell special variable, which has meaning as the current PID on Linux/AIX platforms. To use a global variable in the Testing Tool on Linux/AIX platforms, escape the `$$` using one of these forms. For example:

- `hciroutetest -a '$$variablename'`
- `hciroutetest -a \$$variablename`

On Windows, use `hciroutetest -a $$variablename`.

hciroutetest Tcl procs

Tcl procs pre-defined for `hciroutetest` are:

- `hciroutetestshowbydest`
Prints the message content to `stdout` by `dest msgcontent`.
- `hciroutetestsavebydest`
Saves the successfully handled message content to the file naming `base.dest`.
- `hciroutetestsavebydeststatus`

Saves the successfully handled message content to the file naming `base.srcid.trxid.dest.0`.

`0 = "success"`

- `hciroutetestsavebydeststatus _errtps`

Saves any message which fails in `xlate/route` to the file naming `base.srcid.trxid.dest.1`.

`1 = "failure"`

- `hciroutetestsavedbydisp`

Saves the message to a file.

- `hciroutetestshowbydisp`

Displays the message in the IDE.

There is also an option to display only the final output message or both the intermedia chained or branched message and the final output message. When this option is selected, the message is displayed after each `xlate`.

When the route test option (`-n show intermedia branched and chained messages`) is selected, `hciroutetestsavedbydisp` OR `hciroutetestshowbydisp` is called after each `xlate`.

- `hciroutetestsavedbydisp` saves the message to a file.
- `hciroutetestshowbydisp` displays the message in the IDE.

When the route test option (`-n show intermedia branched and chained messages`) is selected and **Send To Proc** is `hciroutetestshowbydest`, the message between the `xlate` and the destination is displayed. For example:

```
test1.xlt msg
test2.xlt msg
dest msg
```

When the route test option (`-n show intermedia branched and chained messages`) is selected and **Save to File** is selected, then the message between the `xlate` and destination is saved to the selected file.

For example:

- 1 Save the file to `aaa`.
- 2 Run `hciroutetest`.

Files `aaa`, `aaa.branch_dest1`, `aaa.branch_dest2` and messages after `test2.xlt` and `test3.xlt` are saved to file `aaa`.

Messages after `test3.xlt` and the `branch_dest1` message are saved in file `aaa.branch_dest1`.

Messages after `test1.xlt` and the `branch_dest2` are saved in file `aaa.branch_dest2`.

hcischedulercctl

The Scheduler is a Linux/UNIX service that can start automatically when the system starts.

On Windows, the Scheduler is run as a stand-alone process, and starts when Cloverleaf Integration Services is started.

You can use `hcischedulercctl` to start/stop/restart the Scheduler at any time.

Usage:

```
hcschedulerctl [-s] | [-k] | [-r] | [-l] | [-h|--help]
```

- `-s` starts the Cloverleaf scheduler.
- `-k` stops the Cloverleaf scheduler.
- `-r` restarts the Cloverleaf scheduler.
- `-l` shows the job status within the Cloverleaf scheduler.
This lists the current site's job status if a site is set; otherwise, it lists all job statuses within `hciroot`.
- `-h` shows the help.

For additional information, see [Scheduler](#) on page 110.

hciscriptedit

This command edits Tcl scripts and is identical to `hciprocedit`.

```
hciscriptedit [-gui] [-t type] script-file
```

- `-gui` starts an Xterm window.
- `-t type` specifies the type of the script file, where *type* is:
 - `tps` specifies the TPS module.
 - `trxid` specifies the transaction ID determination procedure.
 - `xltp` specifies the translation procedure that is called from within an xlate code fragment.
- *script-file* is the base name of the file to edit. `.tcl` is added if this is not present.

hcisecupgrade

Usage:

```
hcisecupgrade [-mode mode] -cacertpath path -caname name  
-capass password -hostpass password -adminpass password  
-sshhostname host [-ssport port] [-l log]
```

- `-mode mode`
- `-cacertpath path` is the CA public certificate location.
- `-caname name` is the CA certificate name.
- `-capass password` is the CA password.
- `-hostpass password` is the host server password.
- `-adminpass password` is the default user administrator password.
- `-sshhostname host` is the security server host name.
- `-ssport port` is the security server port.
- `-l log` is the log file path.

hcisecupgrade mode options

Usage:

```
hcisecupgrade [-mode mode] [args]
```

The available `-mode` options are:

- `NONE_TO_ADVANCED`
This is the default when no mode is specified.
- `NONE_TO_BASIC`
- `ADVANCED_TO_NONE`
- `BASIC_TO_NONE`

Return code

After a command is run, there is a return code:

- 0 indicates success in changing the security mode.
- 1 indicates a failure to change the security mode.

Command help

Use `hcisecupgrade -H` for all available modes.

Use `hcisecupgrade -mode mode -H | -help` for usage of a specified mode.

hcisecupgrade: None to Advanced

This upgrades security from None to Advanced. This is the default when no mode is specified.

Usage:

```
hcisecupgrade [-mode NONE_to_ADVANCED]-cacertpath path  
-caname name -capass password -hostpass password  
-adminpass password -sshhostname host [-ssport port]  
[-l log]
```

- `-mode NONE_to_ADVANCED`
- `-cacertpath path` is the CA public certificate location.
- `-caname name` is the CA certificate name.
- `-capass password` is the CA password.
- `-hostpass password` is the host server password.
- `-adminpass password` is the default user administrator password.
- `-sshhostname host` is the security server host name.
- `-ssport port` is the security server port.
- `-l log` is the log file path. The default value is `%HCIR00T%/server/logs/hcisecupgrade.log`.

hcisecupgrade: None to Basic

This upgrades security mode from None to Basic.

Usage:

```
hcisecupgrade [-mode NONE_to_BASIC] -cacertpath path  
-caname name -capass password -hostpass password  
-adminpass password [-l log]
```

- `-mode NONE_to_BASIC`
- `-cacertpath path` is the CA public certificate location.
- `-caname name` is the CA certificate name.
- `-capass password` is the CA password.
- `-hostpass password` is the host server password.
- `-adminpass password` is the default user administrator password.
- `-l log` is the log file path. The default value is %HCIR00T%/server/logs/hcisecupgrade.log.

hcisecupgrade: Advanced to None

This downgrades security mode from Advanced to None.

Usage:

```
hcisecupgrade [-mode ADVANCED_to_NONE] -capass password  
[-l log]
```

- `-mode ADVANCED_to_NONE`
- `-capass password` is the CA password.
- `-l log` is the log file path. The default value is %HCIR00T%/server/logs/hcisecupgrade.log.

hcisecupgrade: Basic to None

This downgrades security from Basic to None.

Usage:

```
hcisecupgrade [-mode BASIC_to_NONE] -capass password  
[-l log]
```

- `-mode BASIC_to_NONE`
- `-capass password` is the CA password.
- `-l log` is the log file path. The default value is %HCIR00T%/server/logs/hcisecupgrade.log.

hcisecurityaudit

The `hcisecurityaudit` command supports security audit result data generation for a specific site and root.

When this command is run, it first checks the license.

```
hcisecurityaudit [-t site1,site2,...] [-e [-d export report path]
[-f]] [-l] [-h | -help]
```

- `-t site1,site2,..` is the site name list. For root, this is `HCIRoot`. Without this option, the root and all sites are scanned.
- `-e` merges all scan results of the specific sites as a plain text file.
- `-d export report path` specifies the file path to which to export the security audit report.
If the path is a directory, then the default file name is `securityaudit_report.txt`.
- `-f` forces an overwrite of the export file.
- `-l` prints the scan process information to the log file.
- `-h` or `-help` displays this help message.

hcisetenv

Specifies changes to the user environment. This command is run when users run the `setroot` and `setsite` commands.

Note: You must select one of `-remote`, `-root`, or `-site`.

You can use any of these commands:

```
hcisetenv -remote rootdir [sitename]
```

```
hcisetenv -root targetshell [rootdir
[sitename]] hcisetenv -root -clear
```

```
hcisetenv -site site hcisetenv
-site -clear
```

For these commands:

- `targetshell` specifies the shell the script uses to run, such as `ksh`, `csh`, or `tcl`.
- `-root rootdir` specifies the directory that is the root the setup uses, such as `/cisversion/integrator`.
- `-site site` specifies the site name within the root.
- `-clear` unsets the root or the site.

clgui option

For the GUI to get the same `setsite` variables from `hcisetenv`, the `-clgui` option sends the environment variables to stdout for the GUI to read.

This format has one variable per line. No quotes are necessary.

=

Run the script as this:

```
$HCIR00T/sbin/hcisetenv -site clgui
```

hcisitearchive

This command archives a site directory within the current root in Cloverleaf . When this is finished, it can be restored after doing a scratch install of a newer version of Cloverleaf.

`hcisitearchive` includes the `version` folder. This is for version control.

When a user adds a configuration into version control, a version folder is created in the current site to store the configuration.

Note: This command only runs on UNIX.

hcisitecleanup

This command removes crashed process IDs, `cmd_port` files, `MonitorDs`, and `monitorShmem` files.

Note: All processes must be down before running this command.

hcisiteconvert

This command converts a site. This is called from within `hcirootcopy`.

```
source-sitedir [log]
```

- *source-sitedir* specifies source-site directory.
- *log* specifies the log file. Default value is `siteconvert.log`.

hcisitectl

This command controls the Site Daemons.

```
hcisitectl [-f] [-h host] [{ -K | -k what }]
[{ -S | -s what }] [{ -R | -r what } -d delay interval]
[-u #users] [-A args] [-n] [-v]
```

- `-f` forces stopping of daemons.
- `-h host` is the name of the remote host.
- `-K` stops all daemon processes.
- `-k what` stops specified daemons and *what* is a comma-separated list:
 - `l` is the lock manager.

- `m` is `hcimonitorD`.
- `-s` starts all daemon processes.
- `-s what` starts specified daemons and *what* is comma-separated list:
 - `l` is the lock manager.
 - `m` is `hcimonitorD`.
- `-R` restarts all daemon processes.
- `-r` is similar to `-k` and `-s`.

Use `{-r what}` to restart a specified daemon in a comma-separated list.

- `l` is used for the Lock Manager.
- `m` is used for `hcimonitorD`.
- `-d delay interval` is the unit for delay interval is seconds. This is a pause between the stop and start. The default value is 30s, and is defined in the GUI at the site-level configuration. For a different time-out, use the command line directly.

For further information, see [hcieengineerestart..](#)

- `-u #users` specifies the maximum number of users for lock manager. `#users` defaults to 500 and must be greater than 200.
- `-A args` specifies the startup args for daemons and *args* is a comma-separated list. Each entry is of the form *what=args*, where *what* is `d`, `l`, or `m` and *args* are the arguments to pass to that daemon.
- `-n` indicates not to run under service on NT.
- `-v` specifies to do monitorD duplicate checks when starting/stopping monitorD.

If no options are specified, then the status of the daemons is reported.

Note: Do not stop the Lock Manager without first ensuring that there are no running engine processes in the site.

Order of running

The `errdb` alert type requires database access, so during start-up, the Lock Manager should be started before starting the Monitor Daemon.

For shutdown, the Monitor Daemon should be shut down first, followed by the Lock Manager.

Starting all daemons

Use the `-s` argument to start all the daemons.

Starting a specific daemon

To start a specific daemon with an alert:

```
hcisitectl -s m -A "m=-cl foo.alert"
```

In this example, the alert file is named `foo.alert`.

For MonitorD, the available parameters are the same as `hcimonitorD`:

- `-de eoc-pattern` enables an engine output configuration pattern.
- `-dd eoc-pattern` disables an engine output configuration pattern.
- `-cl cfg-list` specifies a list of alert configuration files to process. Use a space to separate multiple file names in the list. If no file name is specified, then `default.alert` is processed, if it exists.
- `-nl` enables non-daemon mode, where there is no log file. In this case, the Monitor Daemon runs in the foreground. This option is only used for debugging.
- `-D` enables debugging information.
- `-S sitename` is the site name.

Starting a specific alert

Use this format to run a specific alert. In this example, the alert file is named `foo.alert`:

```
hcisitectl -s a -A "a=-cl foo.alert"
```

hcisitedoc

This command generates HTML pages of the specific sites. The command parses all available resource files in the specified sites and produces a corresponding set of HTML pages.

```
hcisitedoc -s site_name [site2_name ... [siten_name]
[-d destination_dir] | [-h | -help]
```

- `-s site_name [site2_name ... [siten_name]` specifies the sites for which the document is generated.
- `-d destination_dir` specifies in which folder the document is generated.
- `-h` or `-help` shows the help usage.

If `destination_dir` is not specified, then the site doc is generated under `$HCIR00T/server/tomcat/webapps/sitedoc`.

After the command line is issued, `stdout` prints the detailed generating processes. For example:

```
Generating Site Document for Site XXX...
Generating Netconfig...
Generating HL7...
    Generating variant YYY...
    Generating variant ZZZ...
```

hcisiteinit

This command creates a new site.

```
hcisiteinit [-c site] [-s] [-i] [-p path] [-E SMATDBFlag]
[-D type] [-K SMATDBKey] [-I InternalDBFlag] [-M InternalDBKey] [-R ErrDBFlag]
[-N ErrDBKey] newsite
```

- `-c site` is the site to be cloned by *newsite*.
 - If `-c site` is used and no flags are used, then database encryption stays consistent with the configuration in the template site.
 - If `-c site` is not used and no flags are used, then the databases are encrypted by default and use the default key.
- `-s` runs `hcisitectl -s` after site creation.
- `-i` ignores restrictions on the site name.
- `-p path` is the target path where *newsite* is saved.
A symbolic link `$HCIRoot/newsite` is created linking to `<path>/newsite` after site creation. This is available only in NTFS on Windows. The target path must be R/W to the user.
- `-E SMATDBFlag` encrypts/decrypts SMAT.
 - `1` indicates encrypted. This is the default.
 - `0` indicates unencrypted.
- `-D type` is the driver type that is used to read/write SMAT. Acceptable values are `file` and `sqlite`. When `-D type` is `sqlite`, the `sqlite` database retains the same behavior as the internal and error databases.
- `-K SMATDBKey` is the encryption key of the SMAT database.
- `-I InternalDBFlag` is the internal database. This flag has a higher priority than the encryption setting in the template site specified by `-c site`.
 - `1` indicates encrypted. This is the default.
 - `0` indicates unencrypted.
- `-M InternalDBKey` is the encryption key of the internal database.
- `-R ErrDBFlag` is the error database. This flag has a higher priority than the encryption setting in the template site specified by `-c site`.
 - `1` indicates encrypted. This is the default.
 - `0` indicates unencrypted.
- `-N ErrDBKey` is the encryption key of the error database.
- *newsite* is the name of the new site to create. This must not currently exist.

hcisiterestore

This command untars the package file from `hcisitearchive` and then restores the site that was archived using `hcisitearchive`.

This includes the version folder that was archived by `hcisitearchive`.

When a user adds a configuration into version control, a `version` folder is created in the current site to store the configuration.

Note: This command runs only on UNIX.

hcismat

```
hcismat -i file_name operation -pri priority -nl -reply -ib
message selection criteria
```

Note: One of `-orut`, `-oruf`, `-orst`, `or-orsf` and `-i` options are required.

- `-i file_name` is the input file name.
- `-pri priority` is the resend priority (4096 to 8192). The default is 5120.
- `-reply` resends selected messages as reply instead of data.
- `-ib` resends selected messages as inbound instead of outbound.
- `-nl` writes the file as newline delimited. The default is `len10`.
- `operation` is exactly one of:
 - `-orut thread` resends unselected messages to thread `thread`.
 - `-oruf file_name` resends unselected messages to file `file_name`.
 - `-orst thread` resends selected messages to thread `thread`.
 - `-orsf file_name` resends selected messages to file `file_name`.
- `message selection criteria` is any valid grouping of:
 - `-sds date` starts at date and time `date yyyyymmdd[hhmmss]`.
 - `-sde date` ends at date and time `date yyyyymmdd[hhmmss]`.
 - `-sdn` Notes the meaning of the date search.
 - `-sms mid` starts at message ID `mid[0.0.]###`.
 - `-smemid` ends at message ID `mid[0.0.]###`.
 - `-smn` Notes the meaning of the `mid` search.
 - `-sall` selects all messages in the input file.

Note: SQLite is integral to engine functionality. When the first connection to a `smatdb` file is opened, `wal` and `shm` files are temporarily created. These files are located in the same directory as the `smatdb` file and use the same name as the `smatdb` file. These are normally removed when the last connection to the `smatdb` file is closed. If you move the `smatdb` file, then you must also move these temporary files. Otherwise, data is lost.

hcismatconvert

The current SMAT files are not compatible with earlier versions of the GUI. The `hcirootcopy` command-line tool does not automatically convert old SMAT index files to the new format, as the GUI continues to support the old format.

It does, though, give you a message that `hcismatconvert` is available for converting. Because `hcirootcopy` uses `hcisiteinit` to create the new sites, the SMAT history default settings are written into the new `siteInfo` file.

`hcismatconvert` is a conversion tool where you can convert SMAT files between formats: old to new, or new to old.

Converting to the new format, it adds/subtracts white space as necessary to meet fixed-length standards:

- New fields in the index are given null or predetermined default values. These match the metadata that the message would have received upon resend in the old implementation of SMAT.
- Variable-length metadata is given zero length, so no changes are necessary to the message file.

Note: The default is a forward conversion that overwrites the input file. If the input file name has the `.smatdb` extension, then this is a SMAT database export.

```
hcismatconvert [-b] [-f] input file name [-o output file name]
               [-s siteInfo path]
```

- `-b` is a backwards conversion, from new to old. This cannot be used for SMAT database import/export.
- `-f` forces a database update. This is only for a database import.
 - If the output file exists, then the database import fails.
 - If `-f` is specified, then the data is imported to the existing database of that name.
- *input file name* is the input file named *file name*.
- `-o output file name` outputs to the file named *output file name* instead of overwriting input.
- `-s siteInfo path` is the `siteInfo` path for the SMAT database import/export. This cannot be used for a file database export.

SMAT database file

There are two formats for file-based SMAT: old and new. This script converts between the file-based SMAT format and database-based SMAT format. This script can only be used as a database export utility.

When a SMAT database file is encrypted, the encryption information is read from `$HCISITEDIR/siteInfo` environment variable or it can be overridden with the `-s` option.

Example: Backward conversion

In this example, `conn_1.msg` and `conn_1.idx` are SMAT files of the new format.

You then use this command:

```
hcismatconvert -b conn_1 -o conn_1_old
```

This takes the contents of `conn_1.msg` and `conn_1.idx`, converts them to the old format, and writes them to files named `conn_1_old.msg` and `conn_1_old.idx`.

To overwrite, use:

```
hcismatconvert -b conn_1
```

Using this command would do the same conversion, but overwrite `conn_1.msg` and `conn_1.idx`.

Example: Forward conversion

In this example, `conn_2.msg` and `conn_2.idx` are files of the old SMAT format.

You then use this command:

```
hcismatconvert conn_2 -o conn_2_new
```

This takes the contents of `conn_2.msg` and `conn_2.idx`, converts them to the new format, and writes them to files named `conn_2_new.msg` and `conn_2_new.idx`.

Metadata fields not stored in the old format are given default values. Note that the two `.msg` files are identical because variable-length metadata would default to length zero.

When `-o` is specified, the `.ecd` file is also copied to the output file name.

`hcismatconvert conn_2` would do the same conversion, but overwrite `conn_2.idx`; `conn_2.msg` would remain unchanged.

Example: Converting a SMAT database file to a flat file

In this example, `conn_3.smatdb` is the SMAT database file.

You then use this command:

```
hcismatconvert conn_3.smatdb -o conn_3
```

This takes the contents of `conn_3.smatdb`, converts it to a flat file format, and writes it to files named `conn_3.msg` and `conn_3.idx`.

Example: Converting a file-based SMAT history to database-based SMAT

To do this, use a file-based SMAT as the input file and specify a database-based SMAT file as the output file.

For example, to convert the file-based SMAT files `file.idx`, `file.msg`, and `file.ecd`, use:

```
hcismatconvert file -o dbConverted.smatdb
```

This command locates the files with rootname `file` and then finds the `file.idx`, `file.msg`, and `file.ecd` files.

For the output, it creates `dbConverted.smatdb` and `dbConverted.ecd`.

hcismatdbencrypt

To disable SMAT database encryption:

```
hcismatdbencrypt -s site_name -d | [-h | -help]
```

- `-s site_name` specifies the site.
- `-d` disables encryption.
- `[h | -help]` displays this help message.

Return codes are:

- 0: Command line succeeded.
- 1: Command line failed.

hcismatcrypt

The SMAT database encryption key (SMATDB key), can be customized. This key is stored in `siteInfo` in the format `smatdbkey=XXX`.

It can be customized/modified when creating a site or changing the SMATDB key from the GUI. The `hcismatcrypt` tool is provided to validate/change the encryption key of a SMAT DB file.

Generally, all encrypted SMATDB files under a site share the same `smatdbkey` in `siteInfo`. If one SMAT database file does not have the same SMATDB key, then the GUI requires specifying the correct key when searching it. Otherwise, it is ignored when changing the SMATDB key.

```
hcismatcrypt command [args]
```

command is one of:

- `validatekey` validates the encryption key.
- `rekey` changes the encryption key.

For usage of a specific command, use:

```
hcismatcrypt command -H
```

hcismatcycle

This command shows

```
hcismatcycle [-s sitename] [-p processname]  
[-t threadname] [-z] [-b in|out] [-r HCIR00T] -h
```

- `-s` is the site name.
- `-p` is the process name.
- `-t` is the thread name.
- `-z` is the MSI reset on `-t threadname`.
- `-b` is inbound or outbound.
- `-r` is the absolute path of a specified `HCIR00T`.
- `-h` is the usage help.

hciss

This command starts, stops, and monitors a system server.

It requires `HCIRoot` to be set in the environment. If this is not set, then an error message is displayed.

For help, use one of these:

```
hciss -h
```

```
hciss -help
```

Note: `hciss` runs only locally on the same host on which the host server or security server is installed.

Stopping a server

Use `hciss` to stop a host server or security server. When shut down, the appropriate `server.log` or `security.log` shows a terminated message.

Host server:

```
hciss -k h
```

Security server:

```
hciss -k s
```

Starting a server

Use `hciss` to start a host server or security server. This is the suggested method.

After starting the host server or security server with `hciss`, bringing down the service shuts down the host server or security server. This is in contrast to using `hcihostserver.exe` and `hcisecurityserver.exe`, which have no contact with the Windows service.

Host server:

```
hciss -s h
```

Security server:

```
hciss -s s
```

Note: You can use the `hciss` command to check server status without going to Task Manager in Windows or `ps` in UNIX.

Monitoring a server

Use `hciss` to check the status of a host server or security server, running or non-running.

hcitblconvert

This command upgrades the tables in the table directory to the current and latest version.

To convert all tables, use:

```
hcitblconvert
```

This converts all tables in the table directory and saves the old tables with a .bak extension.

To convert a specific table, use:

```
hcitblconvert tablename
```

This converts a specific table in the table directory and saves the old table with a .bak extension.

hcitbls

This command lists all table lookup configuration files in a site.

You can use one of these commands:

```
hcitbls [-lp]
```

```
hcitbls [-l] [-p]
```

-l shows the long listing.

-p shows file prologues.

hcitcl

This command starts the interactive Tcl shell. This is useful for checking Tcl command syntax.

`msiDetach` cannot be used in UPoC in the runtime of engine. It detaches MSI shared memory and breaks access to shared memory in the runtime of engine. It can, though, be used in the `hcitcl` interface (for non-UPoC scripts) as shown in:

```
hcitcl>msiAttach
hcitcl>msiDetach
[0:TEST] Msi shared memory is detached!
hcitcl>msiAttach
hcitcl>msiAttach
Error: shared memory region is already attached
hcitcl>msiTocEntry
conn_1 dest_1
hcitcl>msiGetStatSample
Error: wrong # args: msiGetStatSample <threadName | index> ?retvar?
hcitcl>msiGetStatSample conn_1
{ALIVE 0} {LASTEXTRACT 1257762619} {LASTUPDATE 1257738797} {START 1257738697} {STOP 1257738797}
{PSTATUS down} {PFLAGS 0} {PLASTREAD 1257738701} {PLASTWRITE 1257738716} {PLASTERROR 0} {PLAS
TERRTEXT {}} {XLATECNT 487} {FORWARDCNT 0} {ERRORCNT 0} {MSG SIN 480} {MSG SOUT 334} {BYTESIN 9789}
```

```
{BYTESOUT 6969} {IBLATENCY 13036.640} {OBLATENCY 271.431} {TOTLATENCY 331.476} {IBPRESMSQD 0}
{IBPOSTSMSQD 0} {OBPRESMSQD 0} {OBDATAQD 0} {OBREPLYQD 0} {INTERTHREAD {{{NAME conn_1}{NSENT 0}
{NRCVD 334} {POSTXLTQD 0} {XLATETIME 0.000} {TIMEONQ 240.344} {TOTLATENCY 331.476}}} {{{NAME
dest_1} {NSENT 487} {NRCVD 0} {POSTXLTQD0} {XLATETIME 12753.589} {TIMEONQ 0.000} {TOTLATENCY
0.000}}}}}
hcitcl>msiAttach
Error: shared memory region is already attached
hcitcl>msiDetach
[0:TEST] Msi shared memory is detached!
hcitcl>msiDetach
Error: shared memory region is already detached
hcitcl>
```

hcitcptest

This command helps debug TCP/IP interfaces.

```
hcitcptest [-n] [-h hostname] [-f file] -t tcp_encoding_type
-p port
```

- `-n` specifies to run in non-interactive mode, and does not read from stdin. This should be used only in server mode.
- `-h hostname` specifies an optional host name. If *hostname* is not specified, then `hcitcptest` works as a server that listens on the port that is specified by the `-p port` option. Use this argument to configure as server or client.
- `-f file` specifies an optional file name. All data received from the host is written to this file, including any encoding characters.
- `-t tcp_encoding_type` specifies encoding type:
 - `2` specifies 2-byte length-encoding.
 - `2e` specifies 2-byte length-encoding, exclusive of encoding.
 - `4` specifies 4-byte length-encoding.
 - `4e` specifies 4-byte length-encoding, exclusive of encoding.
 - `orsos` specifies ORSOS 2-byte length-encoding.
 - `m1p` specifies HL7 MLP encoding.
 - `raw` specifies data is sent as-is with line feed.
 - `n1f` specifies data is sent as-is without line feed.
- `-p port` specifies the port on which to listen or connect.

hcitpsconvert

This command promotes TPS scripts to the latest system version.

```
hcitpsconvert directory
```

directory specifies the directory that contains the Tcl procs that require conversion. Because the directory is specified, this command does not necessarily depend on a `setroot/setsite` command.

If a directory parameter is not specified, `hcitpsconvert` goes to the `tclprocs` directory of the current site. If you run a `setroot/setsite`, it converts the Tcl procs in `$HCISITEDIR/tclprocs`.

hcitptest

This command tests user-created Tcl scripts in various contexts.

```
hcitptest [-a] [-x encoding] [-c caller]
[-C script] [-C tclscript] [-e "proc args"]
[-f format] [-w search expression] [-i interval] [-L]
[-m count] [-p dbgport] [-P portNumber] [-r runmode] [-S]
[-s savebase] [file] ["proc1 [args1]"
..."procN [argsN]"]
```

- `-a` processes all records in the file.
- `-x encoding` is the encoding name for the data file.
- `-c caller` specifies the caller context name.
- `-C script` is the flag that starts the debugger on this script.
- `-C tclscript` is a user-specified script that is evaluated before running the UPoC. This is only supported for Tcl UPoC.

You can use the script to do configurations. For example, to load external Tcl files using the `source` command:

```
-C 'source myproc.tcl'
```

- `-e proc args` specifies to end processing configuration.
- `-f format` specifies the format type:
 - `-f len10` specifies 10-byte length-encoded.
 - `-f nl` specifies newline terminated. This is the default. search expression
 - `-f eof` specifies end-of-file terminated.
- `-w string` specifies the search string or regular expression for which to search.
- `-i interval` specifies the delay interval. This is time mode.
- `-L` specifies Tcl handle leak detection.
- `-m count` specifies the maximum message count. This is time mode.
- `[-P dbgport]` is the TCP listening port number of the debugger server. If this is not given, then a free port is allocated by the operating system.

Example:

```
hcitptest -P 6688 -C "source C:/cloverleaf/cis6.2/integrator/tclprocs/tpsTestHL7Parse.tcl"
-r run -x ASCII -f nl -c sms_ib_data -e "hcitptestshowbydisp" C:/cloverleaf/cis6.3/integra
tor/test/
data/hl7_patient.dat "tpsTestHL7Parse"
```

- `[-p portNumber]` search indicates the listening port for debugging.
- `-r runmode` specifies the run mode:
 - `-r run` specifies normal run mode.
 - `-r start` specifies startup mode.
 - `-r time` specifies time-based mode.
 - `-r shutdown` specifies shutdown mode.
- `-s` specifies to run message-less startup.
- `-s savebase` specifies Save File basename.

- *file* specifies the data file with which to test. This is run mode.
- *procN argsN* specifies TPS module *procs* and *args*.

Note: To specify a file location for test output, you can use "/" as the file delimiter. This applies to all platforms. In Windows, you can also begin the file path with "\\\".

Global variable support

Global variable support is available for the `hcitptest` command line tool. The global variable indicator `$$` has a conflict with the shell special variable, which has meaning as the current PID on Linux/AIX platforms. To use a global variable in the Testing Tool on Linux/AIX platforms, escape the `$$` using one of these forms. For example:

- `hcitptest -a '$$variablename'`
- `hcitptest -a \$$variablename`

On Windows, use `hcitptest -a $$variablename`.

Contexts

The available contexts are:

- `ack_control`
Protocol ACK/NAK control.
- `fileset_ibdel`
If the Fileset FTP driver is set to `local-tps` or `FTP`, then this context can override the protocol driver's default setting. The oldest is deleted first. It also programmatically controls which files are deleted after processing.
- `fileset_ibdirpars`
If the Fileset FTP driver is set to `local-tps` or `FTP`, then this context can override the protocol driver's default setting. The oldest is read first. It also programmatically controls when files are read in by the driver.
- `pduppoc_read`
Instructions that are provided for the UPoC driver when it performs a read.
- `pduppoc_write`
Instructions that are provided for the UPoC driver when it performs a write.
- `prewrite`
Instructions that are provided for the UPoC driver before it performs the write.
- `proto_startup`
Protocol startup.
- `proto_statup_sendfail`
Notification that is invoked upon an unsuccessful protocol startup.
- `proto_startup_sendok`
Notification that is invoked upon a successful protocol startup.

- `reply_gen`
Reply generation that is invoked after reply time-out.
- `send_data_ok`
Notification that is invoked upon successful data message send.
- `send_data_fail`
Notification that is invoked upon failed data message send.
- `send_reply_ok`
Notification that is invoked upon successful reply message send.
- `send_reply_fail`
Notification that is invoked upon failed reply message send.
- `sms_fwd_data`
TPS data message forwarding stack.
- `sms_fwd_reply`
TPS reply message forwarding stack.
- `sms_ib_data`
TPS IB (inbound) data message stack.
- `sms_ib_reply`
TPS IB reply message stack.
- `sms_ob_data`
TPS OB (outbound) data message stack.
- `sms_ob_reply`
TPS OB reply message stack.
- `trixd`
Transaction ID that is used for translation and routing.
- `xlt_gen`
Generate procedures for route detail.
- `xlt_post`
Xlate post-processing.
- `xlt_pre`
Xlate pre-processing.
- `xlt_raw`
Xlate raw processing.
- `httpc_query`
- `dtc_action`
User-defined action TPS that is configured for every DTC state.
- `dtc_rollback`

hcitrxidtest

This command simulates getting the TRXID on an incoming message from the source thread.

This runs any inbound TPS for the source thread before the engine retrieves the TRXID. This is necessary because the TPS could change the message content.

```
hcitrxidtest [-a] -C [tclscript] [-V] [-x encoding] [-d] [-f format]
[-c drivercontrol] -P [dbgport] [-r thread] thread data_file
```

- `-a` processes all records in the file.
- `-C [tclscript]`
Specify a script to evaluate before running the UPoC. This is only supported for Tcl UPoC.
You can use the script to do configurations. For example, to load external Tcl files using the `source` command:


```
-C 'source myproc.tcl'
```
- `-v` shows verbose mode.
- `-x encoding` is the encoding name for the data file. This encodes from *encoding* to UTF-8.
- `-d` processes as DATA messages. This is the default.
- `-f format` specifies the file format:
 - `-f len10` specifies 10-byte length-encoded.
 - `-f nl` specifies newline terminated. This is the default.
 - `-f eof` specifies end-of-file terminated.
- `-c drivercontrol` is used to set the content to message metadata DRIVERCONTROL. You can input any content to set to *drivercontrol* using this option.
For a test on DICOM messages, `-c drivercontrol` is required, where *drivercontrol* is {AbstractSyntax***} {TransferSyntax***}. This is required to get the TRXID.
See [DICOM abstract syntax values](#) and [DICOM transfer syntax values](#).
- `-P [dbgport]`
This is the TCP listening port number of the debugger server. If this is not given, then a free port is allocated by the operating system.
- `-r thread` processes as REPLY to thread.
- *thread* specifies the source thread to simulate.
- *data_file* specifies the data file with which to test.

hciunixservice

This command adds/removes auto-starting of `hciss` on UNIX/Linux systems.

You can use one of these commands:

```
hciunixservice -i h|s [auto-start script directory] [-c]
```


or

```
or: hciunixservice -r h|s
```

- `-i` installs the specified services.
Additional auto-start scripts install paths can be assigned, under which the Cloverleaf's auto-start service is referred
- `-r` removes the specified services.
- `h` is the host server service.
- `s` is the security server service.
- `-c` is a special option for CentOS 7 and above. This tunes `Systemd` unit files.
By default, assertions are made on `HCIRoot` for services. An error is reported to the system to mark the service as a failure.
Under a cluster environment, `HCIRoot` is only available to the backup node when the major one is down. This option can be used to mute alerts. `Systemd` skips the service when `HCIRoot` is absent.

Note: The command must act as `root` to work.

This command resides in `$HCIRoot/sbin/`.

Example

To add/remove auto-starting of `hciss` on UNIX/Linux systems.

```
hciunixservice [-i h|s] | [-r h|s]
```

- `-i` installs the specified services.
- `-r` removes the specified services.
- `h` is the host server.
- `s` is the security server.

To install the host server as a service:

- 1 Log in to root:
`su` OR `sudo` to root.

- 2 Change the directory:

```
cd $HCIRoot/bin
```

- 3 Install the host server as a service:

```
hciunixservice -i -h
```

- Replace `-i` with `-r` to remove the service.
- Replace `h` with `s` to install/remove the security server as a service.

- 4 Start the service:

```
hostserver service start
```

When you shut down the system, the host/security server and processes, if assigned, gracefully shut down and auto-start after the system boots.

hciupdateallowlisthash

Use this command to update all hash values of a customized allowlist in the specified site.

This impacts the hash values of the command-line allowlist and Java allowlist.

Usage:

```
hciupdateallowlisthash -s site [-h | -help]
```

Where:

- `-s site` specifies a site. This is required.
- `-h` or `-help` displays this help message.

hciupdateencrypt

The Cloverleaf GUI encrypts passwords when saving artifacts. This utility is used to update the encryption method after an upgrade.

A command-line option is available to automatically update the NetConfig, `default.alert`, and lookup table files. Re-saving these artifacts converts the password-protected encoding method to the encrypted method.

This utility uses the CLAPI to open and save the Cloverleaf artifacts.

Note: You must have basic security installed to use this utility.

Before using this command, use `hcirootcopy` to promote sites.

```
hciupdateencrypt { -u username -p password } {-h | -a | -n | -l |  
-s | -d | -g | -t all | -t tablename}
```

- `-u` is your organization's Cloverleaf Wizard user name. This is required.
- `-p` is your organization's Cloverleaf Wizard password. This is required.
- `-h` shows this help message.
- `-a` updates the `default.alert` file.
- `-n` updates the site's NetConfig file.
- `-l` verifies a username and password.
- `-s` updates the `siteInfo` file.
- `-d` updates the `dbconfiguration.ini` file, if it exists.
- `-g` updates the site's NetConfig file, all alert files, `siteInfo` file, `dbconfiguration.ini` file, and all lookup tables.
- `-t tablename` updates the table with the provided file name.
- `-t all` updates all tables found in `sitedir/Tables`.

hcverify

Performs a system validation check.

```
hcverify [-R] [-W] [-hl7] [-p] [-r dir] [-s site]
         [-u user] [-v] [-w] [-x]
```

- `-R` specifies no system default root verification.
- `-W` specifies to suppress warnings.
- `-hl7` specifies to verify only HL7 environments.
- `-p` specifies to check that expected processes are running well.
- `-r dir` specifies to verify the *root* directory (*dir*) including default.
- `-s site` specifies to verify the *site* within the *root*.
- `-u user` specifies to verify the user.
- `-v` specifies verbose operation.
- `-w` specifies to make warnings fatal.
- `-x` specifies to perform intensive xlate testing.

You must `setroot` into the *root* and *site* to verify for the `-x` option to work. `hcverify` can only be run as the *root* user. If it returns an error about files that have an incorrect mode, then rerun the command with the `-F` option. The phrase "incorrect mode" refers to file permissions.

If you run `hcverify` without any options, then it does not display verification information except for warnings or error messages. To have it display the verify procedure, you must run `hcverify` with the `-v` (verbose) option.

hcversion

This command reports the version history list for the specified configuration file.

```
hcversion [-s site name] -f file path [-v versiontag] [-c] [-h | -help] ]
         [-t]
```

- `-s site name` sets the site name of the site. This is optional.
- `-f file path` specifies the configuration file name with relative path. This path is relative to the site path. The BOX entry's path is relative to the root path.
- `-v version` specifies a version number to show the record.
- `-t tag` is a reserved parameter for the `Tag` action. This is 1 by default.
- `-c` hides the comments in version record.
- `-h` or `-help` displays this help message.

For example:

- When a user does not enter a value for `-f`, this error is given:
No sufficient parameters "file path" provided.
- When a user sets an empty value for `-t`, this error is given: The parameter "tag" must have a value.

Normal output for the specified site and file includes information similar to:

Latest Version	Operation	Operation Time	User
(1.6)	REPLACE	014-11-20 10:21:05	test
History Versions			
Version	Reversion Time	User	Comments
1.1	2014-10-28 11:08:02	administrator	13
1.2	2014-10-28 22:53:57	administrator	323
1.3	2014-10-30 21:35:17	administrator	123
1.4	2014-11-4 16:03:25	administrator	123

hcivrlls

This command lists the VRL configuration files in a site.

You can use one of these commands:

```
hcivrlls [-lp]
```

```
hcivrlls [-l] [-p]
```

-l shows the long listing.

-p shows file prologues.

hcivrltest

This command tests VRL data against actual data.

```
hcivrltest [-a] [-e encoding] [-d n] [-f format]  
[-w search expression] vrl file
```

- -a processes all records in the file.
- -e *encoding* is the encoding name for the data file.
- -d *n* shows data at detail level *n*. The detail range is 0-4, where 0 is raw, unparsed data and 4 is the most detailed.
- -f *format* specifies the file format:
 - -f len10 specifies 10-byte length-encoded.
 - -f nl specifies newline terminated. This is the default.
 - -f eof specifies end-of-file terminated.
- -w *search expression* specifies the search string or regular expression for which to search.
- *vrl* specifies the VRL name to test.
- *file* specifies the data file with which to test.

hcix12test

This command tests X12 records. It handles X12 messages that contain:

- An interchange (ISA..IEA)
or
- A single group (GS..GE)
or
- A single transaction set (ST..SE)

`hcix12test` accepts interchanges, groups, or transaction sets:

- If the message starts with `ST` or `GS`, then it is treated as an individual transaction set or an individual group, respectively. For `ST` and `GS` messages, use default separator characters (*, ~ and \).
- If the message does not start with `ST` or `GS`, then it is treated as an interchange. `hcix12test` handles a message with any number of interchanges, each containing any number of groups, which can contain any number of transaction sets. For interchanges, the separator characters are extracted from the `ISA` segments.
- If using group messages, for example `GS` to `GE`, then create a variant that defines the transaction sets in terms of groups. Then, specify such a variant for the test. The path names for a group are different from the transaction set. The `GS` becomes group zero and the first `ST` is `1(0).0(0).ST(0)`.

Group-defined messages use this structure:

```
GS
{
    ST
    ...
    SE
}
GE
```

If an envelope is used, then it extracts the transaction sets and parses the individual transaction sets. If transaction sets are used, then `hcix12test` processes the message without extracting the transaction sets.

```
hcix12test [-a] [-d n] [-e encoding] [-i] [-f format]
[-V version] [-v variant] [-t test] [-c drivercontrol]
[-n] [-F fieldname] [-w search expression] [-p sepChars]
file
```

- `-a` processes all records in the file.
- `-d n` shows data at detail level *n*. The detail level range is 0-4, where 0 is raw, unparsed data and 4 is the most detailed.
- `-e encoding` is the encoding name for the data file.
- `-i` displays field addresses using index notation.
- `-f format` specifies the file format:
 - `-f len10` specifies 10-byte length-encoded.
 - `-f nl` specifies newline terminated. This is the default.
 - `-f eof` specifies end-of-file terminated.
- `-V version` specifies the X12 version. The default is 003050.
- `-v variant` specifies the X12 variant.

- `-t test` specifies the test to run:
 - `-t numbers` tests Nn number conversions.
 - `-t parse` test parses a message. This is the default.
- `-c drivercontrol` is set the content to message metadata DRIVERCONTROL.
- `-n` shows the field names.
- `-F fieldname` specifies the field name only.
- `-w search expression` specifies the search string or regular expression for which to search.
- `-p sepChars` are the separator characters used for parsing or encoding. This needs to be in a keyed list format.
- `file` specifies the data file with which to test.

hcixlatenotify

This command modifies the translation thread.

```
hcixlatenotify cmd [thread[,thread]]
```

cmd specifies the command to send to the xlate thread, using `hcicmd`.

thread specifies a comma-separated list of threads to use.

hcixltconvert

This command converts all format translation files from an earlier version of the system to the later one.

Adding a translation file name converts a specified format translation file from an earlier version of the system to the later one.

```
hcixltconvert xlatename
```

hcixltls

This command lists all the Translation Configuration files in a site.

You can use one of these commands:

```
hcixltls [-lp]
```

```
hcixltls [-l] [-p]
```

`-l` shows the long listing.

`-p` shows file prologues.

hcixlttest

This command tests a translation configuration against an actual data message.

```
hcixlttest [-a] -C [tclscript] [-e encoding] [-d n] [-i] [-l] [-f format]
[-F format] [-c drivercontrol] -P [dbgport] [-R preprocessList]
[-o postprocessList] [-w search expression] [-p sepChars] [-P portNumber]
[-L] [-n] xlt infile outfile
```

- `-a` processes all records in the file.
- `-C [tclscript]`

This is the flag to start the debugger on this script. Specify a script to evaluate before running the UPoC. This is only supported for Tcl UPoC.

You can use the script to do configurations. For example, to load external Tcl files using the `source` command:

```
-C 'source myproc.tcl'
```

- `-e encoding` encodes from enc to UTF-8.
- `-d` shows data at detail level *n*. The detail range is 0-4, where 0 is raw, unparsed data and 4 is the most detailed.
- `-i` displays field addresses using index notation.
- `-l` prints unreadable chars without changing to hex. This is more readable print.
- `-f format` specifies the input file format:
 - `-f len10` specifies 10-byte length-encoded.
 - `-f nl` specifies newline terminated. This is the default.
 - `-f eof` specifies end-of-file terminated.
- `-F format` specifies the output file format:
 - `-F len10` specifies 10-byte length-encoded.
 - `-F nl` specifies newline terminated. This is the default.
 - `-F eof` specifies end-of-file terminated.
- `-c drivercontrol` is used to set the content to message metadata DRIVERCONTROL. You can input any content to set to *drivercontrol*

For xlate files using DICOM as the Source/Destination format, using this option. `-c drivercontrol` is required, where *drivercontrol* is {TransferSyntax***}.

See [DICOM transfer syntax values](#).

using thisExample:

```
hcixlttest -a -e UTF-8 -d 1 -f eof -c "{TransferSyntax 1.2.840.10008.1.2}"
BulkCopy.xlt c_store.dat
```

- `-P [dbgport]`
This is the TCP listening port number of the debugger server. If this is not given, then a free port is allocated by the operating system.
- `-R preprocessList` is the list of pre-xlate procs.
- `-O postprocessList` is the list of post-xlate procs.

To use in `-R preprocessList` and `-O postprocessList` in a shell window, the proc list should be:

```
procname1 {argument} procname2 {argument}
```

Example:

```
-R "test_single {{WORD |pretest}} test_single {{WORD |pre2}}" -O "test_single  
{{WORD |posttest}} {SEG PV2}} test_single {{WORD |post2}}"
```

- `-w search expression` specifies the search string or wildcard for which to search. Only the lines of output that match the search expression are printed out. The search is case-sensitive.
- `-p sepChars` are the separator characters used for parsing or encoding. This must be in a keyed list format.
- `-L` specifies Tcl handle leak detection.
- `-n` shows the field names.
- `xlt` specifies the name of the `.xlt` file to test.
- `infile` specifies the data file with which to test.
- `outfile` specifies the len10 file for raw CONTINUE messages.

hcxmlcompile

This command updates the OCM file associated with a DTD/Schema/DTD-containing-XML file that has changed. It analyzes the structure of the DTD/Schema/DTD-containing-XML. It also creates an object model of that structure used to process and map the XML within translation.

Note: The base schema that defines a message must have at least one global element declaration.

```
hcxmlcompile -f filename -p package dir [-r root element name] [-o  
output file name] [-v on/off] [-d on/off] [-n on/off]  
[-e on/off] [-L on/off] [-F pretty/none] [-N /target/namespace URI/none]  
[-b] [-l] [-c] [-s] [-h]
```

- `-f filename` is the XML file name. This must end in `.xsd` schema, `.dtd`, or `.xml`.
- `-p package dir` is the package directory where the XML file has been placed.
- `-r root element name` only creates an OCM for this message definition.
- `-o output file name` is the name of the output message definition.
- `-v on/off` turns on/off outbound message validation. The default is *on*.
- `-d on/off` turns on/off outbound message default value insertion. The default is *on*. This is independent of validation and namespace checking.
- `-n on/off` turns on/off outbound message namespace checking during validation. This applies only if validation is not *off*. Validation is *v on* or not specified. The default is *on*.
- `-e on/off` turns outbound message empty node pruning *on* or *off*. The default is *on*.
- `-L on/off` turns on/off using the local namespace as the default namespace.
- `-F pretty/none` has the target XML message encoded with newlines between elements and tabbed whitespace to denote nesting level. The default is *none*.
- `-N /target/namespace URI/none` sets the default namespace. If this is not specified, then the default is automatically set by the compiler.
 - *Uses the Default Namespace in XSD File* indicates without option. This is the default.

- *target* indicates the target namespace.
- *namespace URI* indicates the specified namespace URI.
- *none* indicates no default namespace.
- When *-N* is not used, *auto* is the default value. The XML compiler automatically determines the default namespace.

Example:

```
hcxmlcompile current_compile_args
```

- *-b* adds Unicode BOM to the front of the OCM file.
- *-l* lists all of the possible root elements defined in this file.
- *-c* recompiles the Schema.
- *-s* shows the command line used to compile the Schema. The OCM file name must be given with *-o*.
- *-h* shows this usage help.

hcxmltest

This command tests the XML files ready for translation.

```
hcxmltest [-e encoding] [-a] [-l] [-d n] [-f format]
[-w search expression] [-p packageName] ocm
datafile
```

- *-e encoding* is the encoding name for the data file.
- *-a* processes all records in the file.
- *-l* prints unreadable chars without changing to hex, making the text more readable.
- *-d n* shows data at detail level *n*. The detail level range is 0-4, where 0 is raw, unparsed data and 4 is the most detailed.
- *-fformat* specifies the file format:
 - *-f len10* specifies 10-byte length-encoded.
 - *-f nl* specifies newline terminated. This is the default.
 - *-f eof* specifies end-of-file terminated.
- *-w search expression* specifies the search string or regular expression for which to search.
- *-p packageName* specifies the directory containing the XML definition file.
- *datafile* specifies the data file with which to test.
- *ocm* is the name of the message definition.

hcixslttest

This command tests XSLT files with the specified runtime parameters.

```
hcixslttest [-e encoding] [-F format] [-p runtime_parameters]
[-w search expression] xslt infile [outfile]
```

- `-e encoding` is the encoding name for the data file.
- `-F format` specifies the output file format:
 - `-F len10` specifies 10-byte length-encoded.
 - `-F nl` specifies newline terminated. This is the default.
 - `-F eof` specifies end-of-file terminated.
- `-p runtime_parameters` are the runtime parameters to pass into the XSLT engine, in the form of a keyed list. For example, `{PARAM {value}}`.
- `-w search expression` specifies the search string or regular expression for which to search.
- `xslt` specifies the name of the XSLT file to test.
- `infile` specifies the XML data file with which to test.
- `outfile` specifies the outfile file. This is optional.

Validating XSLT files

The system supports XSLT validation files for HL7 CDA 2.0 and HITSP_C32 (2.5 version). These files are located in the root level XSLT folder at `$HCIRoot/xslt`. They can be used by `hcixslttest` or the engine without any changes. They can also be copied and modified to the site level for special use. This is located at `$HCISITEDIR/xslt` or `$HCIMASTERSITEDIR/xslt`.

You should have a document file to be validated according to a particular specification. Generally, this is an XML file, for example, `document.xml`.

Cloverleaf provides XSLT validation files for standard HL7 CDA R2 and HITSP C3 version 2.5. With these, the engine and GUI can support root level XSLT files. With this, you can directly use these XSLT files under the root level. You can also move them to the site level and modify them for a particular purpose.

XSLT files can be validated from the command line using this command:

```
hcixslttest [-e encoding[-F format] [-p runtime_parameters]
xslt infile [outfile]]
```

.

For example, `hcixslttest -e ASCII CDA_2.0/ccd_iso_pretty.xsl document.xml`.

The test result shows the validation report, indicating any errors or warnings where the `document.xml` file violates the standard specifications.

This procedure can also be used from the GUI:

- 1 Launch the testing tool in the GUI.
- 2 Select the **XSLT** tab.
- 3 Specify the validation files in **XSLT File**. For example, `ccd_iso_pretty.xsl` for HL7 CDA R2.
- 4 In the **Choose Data File** browser, specify the file that needs validation. For example, `document.xml`.
- 5 Specify other options as required.
- 6 Click **Run Command**. The validation report is returned in the Result pane.

ssmigration

In CIS 19.1, the migration tool only supports the 6.1 to 19.1 and 6.2 to 19.1 migration options.

This feature migrates the security server database. User certificates and key files that are on the host server are not handled in the security server migration tool.

To accommodate this, a new `ssmigration` command line is available under `%HCIR00T/clgui/bin`.

Note: In addition to the command line, the `migrategui` command is located in `$HCIR00T/security/ssmigration` for security migration in GUI mode.

```
ssmigration [-srcVersion 6.1|6.2 -srcRoot source root directory
[-logfile log file full path]] [-h | -help]
```

- `-srcVersion 6.1|6.2` is the source product version number. Currently, this is 6.1 or 6.2.
- `-srcRoot source root directory` is the source product root full directory.
- `-logfile log file name` is the file name of the log file, using the full path. If there is no `-logfile` option, then use `%HCIR00T/ssmigration.log`.
- `-h` or `-help` displays this help message.

DICOM abstract syntax values

This table lists the DICOM abstract syntax values:

SOP class name	UID
Verification SOP Class	1.2.840.10008.1.1
Storage Commitment Push Model SOP Class	1.2.840.10008.1.20.1
Media Storage Directory Storage	1.2.840.10008.1.3.10
Modality Performed Procedure Step SOP Class	1.2.840.10008.3.1.2.3.3
Basic Film Session SOP Class	1.2.840.10008.5.1.1.1
Print Job SOP Class	1.2.840.10008.5.1.1.14
Basic Annotation Box SOP Class	1.2.840.10008.5.1.1.15
Printer SOP Class	1.2.840.10008.5.1.1.16
Printer Configuration Retrieval SOP Class	1.2.840.10008.5.1.1.16.376
Basic Color Print Management Meta SOP Class	1.2.840.10008.5.1.1.18
Basic Film Box SOP Class	1.2.840.10008.5.1.1.2
VOI LUT Box SOP Class	1.2.840.10008.5.1.1.22
Presentation LUT SOP Class	1.2.840.10008.5.1.1.23
Media Creation Management SOP Class UID	1.2.840.10008.5.1.1.33

SOP class name	UID
Basic Grayscale Image Box SOP Class	1.2.840.10008.5.1.1.4
Basic Color Image Box SOP Class	1.2.840.10008.5.1.1.4.1
Basic Grayscale Print Management Meta SOP Class	1.2.840.10008.5.1.1.9
CR Image Storage	1.2.840.10008.5.1.4.1.1.1
Digital X-Ray Image Storage - for Presentation	1.2.840.10008.5.1.4.1.1.1.1
Digital X-Ray Image Storage - for Processing	1.2.840.10008.5.1.4.1.1.1.1.1
Digital Mammography X-Ray Image Storage - for Presentation	1.2.840.10008.5.1.4.1.1.1.2
Digital Mammography X-Ray Image Storage - for Processing	1.2.840.10008.5.1.4.1.1.1.2.1
Digital Intra - oral X-Ray Image Storage - for Presentation	1.2.840.10008.5.1.4.1.1.1.3
Digital Intra - oral X-Ray Image Storage - for Processing	1.2.840.10008.5.1.4.1.1.1.3.1
Encapsulated PDF Storage	1.2.840.10008.5.1.4.1.1.104.1
Grayscale Softcopy Presentation State Storage SOP Class	1.2.840.10008.5.1.4.1.1.11.1
Color Softcopy Presentation State Storage SOP Class	1.2.840.10008.5.1.4.1.1.11.2
Pseudocolor Softcopy Presentation Stage Storage SOP Class	1.2.840.10008.5.1.4.1.1.11.3
Blending Softcopy Presentation State Storage SOP Class	1.2.840.10008.5.1.4.1.1.11.4
X-Ray Angiographic Image Storage	1.2.840.10008.5.1.4.1.1.12.1
Enhanced XA Image Storage	1.2.840.10008.5.1.4.1.1.12.1.1
X-Ray Radiofluoroscopic Image Storage	1.2.840.10008.5.1.4.1.1.12.2
Enhanced XRF Image Storage	1.2.840.10008.5.1.4.1.1.12.2.1
CT Image Storage	1.2.840.10008.5.1.4.1.1.2
Enhanced CT Image Storage	1.2.840.10008.5.1.4.1.1.2.1
NM Image Storage	1.2.840.10008.5.1.4.1.1.20
Ultrasound Multiframe Image Storage	1.2.840.10008.5.1.4.1.1.3.1
MR Image Storage	1.2.840.10008.5.1.4.1.1.4
Enhanced MR Image Storage	1.2.840.10008.5.1.4.1.1.4.1

SOP class name	UID
MR Spectroscopy Storage	1.2.840.10008.5.1.4.1.1.4.2
Radiation Therapy Image Storage	1.2.840.10008.5.1.4.1.1.481.1
Radiation Therapy Dose Storage	1.2.840.10008.5.1.4.1.1.481.2
Radiation Therapy Structure Set Storage	1.2.840.10008.5.1.4.1.1.481.3
Radiation Therapy Beams Treatment Record Storage	1.2.840.10008.5.1.4.1.1.481.4
Radiation Therapy Plan Storage	1.2.840.10008.5.1.4.1.1.481.5
Radiation Therapy Brachy Treatment Record Storage	1.2.840.10008.5.1.4.1.1.481.6
Radiation Therapy Treatment Summary Record Storage	1.2.840.10008.5.1.4.1.1.481.7
Radiation Therapy Ion Plan Storage	1.2.840.10008.5.1.4.1.1.481.8
Radiation Therapy Ion Beams Treatment Record Storage	1.2.840.10008.5.1.4.1.1.481.9
Ultrasound Image Storage	1.2.840.10008.5.1.4.1.1.6.1
Raw Data Storage	1.2.840.10008.5.1.4.1.1.66
Spatial Registration Storage	1.2.840.10008.5.1.4.1.1.66.1
Spatial Fiducials Storage	1.2.840.10008.5.1.4.1.1.66.2
Deformable Spatial Registration Storage	1.2.840.10008.5.1.4.1.1.66.3
Segmentation Storage	1.2.840.10008.5.1.4.1.1.66.4
Real World Value Mapping Storage	1.2.840.10008.5.1.4.1.1.67
Secondary Capture Image Storage	1.2.840.10008.5.1.4.1.1.7
Multiframe Single Bit Secondary Capture Image Storage	1.2.840.10008.5.1.4.1.1.7.1
Multiframe Grayscale Byte Secondary Capture Image Storage	1.2.840.10008.5.1.4.1.1.7.2
Multiframe Grayscale Word Secondary Capture Image Storage	1.2.840.10008.5.1.4.1.1.7.3
Multiframe True Color Secondary Capture Image Storage	1.2.840.10008.5.1.4.1.1.7.4
VL endoscopic Image Storage	1.2.840.10008.5.1.4.1.1.77.1.1
Video Endoscopic Image Storage	1.2.840.10008.5.1.4.1.1.77.1.1.1
VL Microscopic Image Storage	1.2.840.10008.5.1.4.1.1.77.1.2
Video Microscopic Image Storage	1.2.840.10008.5.1.4.1.1.77.1.2.1

SOP class name	UID
VL Slide-Coordinates Microscopic Image Storage	1.2.840.10008.5.1.4.1.1.77.1.3
VL Photographic Image Storage	1.2.840.10008.5.1.4.1.1.77.1.4
Video Photographic Image Storage	1.2.840.10008.5.1.4.1.1.77.1.4.1
Ophthalmic Photography 8-Bit Image Storage	1.2.840.10008.5.1.4.1.1.77.1.5.1
Ophthalmic Photography 16-Bit Image Storage	1.2.840.10008.5.1.4.1.1.77.1.5.2
Stereometric Relationship Storage	1.2.840.10008.5.1.4.1.1.77.1.5.3
Basic Text SR	1.2.840.10008.5.1.4.1.1.88.11
Enhanced SR	1.2.840.10008.5.1.4.1.1.88.22
Comprehensive SR	1.2.840.10008.5.1.4.1.1.88.33
Procedure Log Storage	1.2.840.10008.5.1.4.1.1.88.40
Mammography CAD SR	1.2.840.10008.5.1.4.1.1.88.50
Key Object Selection Document	1.2.840.10008.5.1.4.1.1.88.59
Chest CAD SR	1.2.840.10008.5.1.4.1.1.88.65
X-Ray Radiation Dose SR	1.2.840.10008.5.1.4.1.1.88.67
12-lead ECG Waveform Storage	1.2.840.10008.5.1.4.1.1.9.1.1
General ECG Waveform Storage	1.2.840.10008.5.1.4.1.1.9.1.2
Ambulatory ECG Waveform Storage	1.2.840.10008.5.1.4.1.1.9.1.3
Hemodynamic Waveform Storage	1.2.840.10008.5.1.4.1.1.9.2.1
Cardiac Electrophysiology Waveform Storage	1.2.840.10008.5.1.4.1.1.9.3.1
Basic Voice Audio Waveform Storage	1.2.840.10008.5.1.4.1.1.9.4.1
Patient Root Query/Retrieve Information Model - FIND	1.2.840.10008.5.1.4.1.2.1.1
Patient Root Query/Retrieve Information Model - MOVE	1.2.840.10008.5.1.4.1.2.1.2
Study Root Query/Retrieve Information Model - FIND	1.2.840.10008.5.1.4.1.2.2.1
Study Root Query/Retrieve Information Model - MOVE	1.2.840.10008.5.1.4.1.2.2.2
Modality Worklist Information Model - FIND	1.2.840.10008.5.1.4.31
General Purpose Worklist Management Meta SOP Class	1.2.840.10008.5.1.4.32
General Purpose Worklist Information Model - FIND	1.2.840.10008.5.1.4.32.1

DICOM transfer syntax values

This table lists the DICOM transfer syntax values:

Name	UID
Implicit VR Endian: Default Transfer Syntax for DICOM	1.2.840.10008.1.2
Explicit VR Little Endian	1.2.840.10008.1.2.1
Deflated Explicit VR Big Endian	1.2.840.10008.1.2.1.99
Explicit VR Big Endian	1.2.840.10008.1.2.2
JPEG Baseline (Process 1):Default Transfer Syntax for Lossy JPEG 8-bit Image Compression	1.2.840.10008.1.2.4.50
JPEG Baseline (Processes 2 & 4):Default Transfer Syntax for Lossy JPEG 12-bit Image Compression (Process 4 only)	1.2.840.10008.1.2.4.51
JPEG Lossless, Nonhierarchical (Processes 14)	1.2.840.10008.1.2.4.57
JPEG Lossless, Nonhierarchical, First- Order Prediction (Processes 14 [Selection Value 1]): Default Transfer Syntax for Lossless JPEG Image Compression	1.2.840.10008.1.2.4.70
JPEG-LS Lossless Image Compression	1.2.840.10008.1.2.4.80
JPEG-LS Lossy (Near- Lossless) Image Compression	1.2.840.10008.1.2.4.81
JPEG 2000 Image Compression (Lossless Only)	1.2.840.10008.1.2.4.90
JPEG 2000 Image Compression	1.2.840.10008.1.2.4.91
JPEG 2000 Part 2 Multicomponent Image Compression (Lossless Only)	1.2.840.10008.1.2.4.92
JPEG 2000 Part 2 Multicomponent Image Compression	1.2.840.10008.1.2.4.93
JPIP Referenced	1.2.840.10008.1.2.4.94
JPIP Referenced Deflate	1.2.840.10008.1.2.4.95
RLE Lossless	1.2.840.10008.1.2.5
RFC 2557 MIME Encapsulation	1.2.840.10008.1.2.6.1
MPEG2 Main Profile Main Level	1.2.840.10008.1.2.4.100
MPEG-4 AVC/H.264 High Profile / Level 4.1	1.2.840.10008.1.2.4.102
MPEG-4 AVC/H.264 BD-compatible High Profile / Level 4.1	1.2.840.10008.1.2.4.103

Security startup commands

System security ensures that the engine is accessed only by authorized users. Basic security is installed on the host server and configured without a separate security server.

Advanced security uses a separate security server and maintains a much higher level of security. Security server, hereafter referred to as advanced security, is available for an additional charge. Contact your Account Manager for additional information.

This section describes the command-line options that are used for basic security. The appropriate directories are added to the `PATH` variable. These commands can be issued from any directory.

Basic security components are:

- Audit Log. This is available in no security and basic security.
- Certificate Manager. This is available in basic security.

Note: The Certificate Manager can be accessed only from the host server.

Server startup commands

Server startup commands are available only at the host machine and cannot be configured remotely. They are not available from a client-only installation.

`hciss` is used to start, stop, and monitor a system server.

Security GUI startup commands

GUIs are provided to view and configure several security- and audit-related functions. Commands that start the security GUIs are:

- `hciauditlog`
This starts the Audit Log Viewer, which lists security-related events. The events logged vary depending upon the security level, no security or basic security.
- `hcicertmgr`
This starts the Certificate Manager. Electronic certificates provide the method for authenticating users. This command is available only at the host server.

Note: The Certificate Manager tool is available with basic or advanced security.

Security utility startup commands

Utilities are provided to view and modify different security-related functions. These commands are:

- `hcipasswd`

This changes the password that is associated with private key. This command is for users to change their passwords and asks for username, old password, and new password. There is no output from this command.

This command is available at the server and client level.

This command defaults to the last used certificate.

- `hcicrlrefresh`

This starts the utility that enables the security administrator to change the CRL, or Certificate Revocation List.

The CRL Refresh utility can:

- Reinststate a CRL after it has expired.
- Change the CRL expiration date.
- Replace the existing CRL with a new one.

New CRLs are generated from the current contents of the `\server\certs\revoked` folder.

hci commands

hci commands include:

- [Control command](#)
- [Datum commands](#)
- [Files commands](#)
- [GDBM commands](#)
- [GRM commands](#)
- [Lists commands](#)
- [Message commands](#)
- [MSI commands](#)
- [Strings commands](#)
- [Table commands](#)
- [Xlate commands](#)
- [XPM commands](#)
- [HTTP commands](#)

Control command

The `foreach2` command is an expanded version of the standard `foreach` command that accepts two lists to scan simultaneously.

foreach2

```
foreach2 ?-all? var1 list1 var2 list2 body
```

If the lists are not the same length, then the shorter list determines how many times the loop gets run.

If the `-all` flag is specified, then the longer list controls the running of the command. When the shorter list is exhausted, its loop variable is set to the null value during running the loop body.

Counter commands

Counter commands include:

- [CtrCurrentValue](#) on page 1412
- [CtrInitCounter](#) on page 1412
- [CtrNextValue](#) on page 1413
- [CtrResetValue](#) on page 1413

CtrCurrentValue

Retrieves the counter's current value without updating it.

```
CtrCurrentValue tag ?mode?
```

- `tag` is the counter file path.
- `mode` is the file name.

CtrInitCounter

Initializes a counter file with the specified parameters.

```
CtrInitCounter tag ?mode? ?start? ?max? ?reset?
```

- `tag` is the counter file path.
- `mode` is the file name.
- `start` is the initial counter value. The default is 1.
- `max` is the maximum counter value. The default is 999999999.
- `reset` is the reset counter value. The default is 1.

CtrNextValue

Retrieves the counter's current value, incrementing the counter's value and changing to its reset value, if required.

```
CtrNextValue tag ?mode?
```

- `tag` is the counter file path.
- `mode` is the file name.

CtrResetValue

Changes the counter's value to its configured reset value.

```
CtrResetValue tag ?mode?
```

- `tag` is the counter file path.
- `mode` is the file name.

Destroying a counter

A counter is destroyed by removing the counter file.

For example, in UNIX:

```
% rm NEW_CTR.ctr
```

In Windows:

```
del NEW_CTR
```

Datum commands

Datum commands include:

- [datcreate](#) on page 1414
- [datdestroy](#) on page 1414
- [datget](#) on page 1414
- [datlist](#) on page 1414
- [datset](#) on page 1414

datcreate

This returns the new datum handle for use by other commands.

```
datcreate ?VALUE? ?TYPE?
```

- `VALUE` is the string value of the new datum.
- `TYPE` is an hci supported data type. For example, `ai`, `ch`.

This creates a new datum of the specified `VALUE` and `TYPE`. New datum `VALUE` defaults to empty and `TYPE` to `ch`.

datdestroy

This destroys datum handles.

```
datdestroy -list list  
datdestroy datId1 datId2...
```

`-list` contains the datum handles to destroy. Without the `-list` option, `datId1`, `datId2`, and so on, are the individual datum handles to destroy.

This command returns an empty string.

datget

This returns attribute values or attribute keys.

```
datget datId ?key?
```

If you do not specify a `key`, then the object's attribute keys are returned.

If you specify a `key`, then that attribute's value is returned.

datlist

This returns a list of valid datum handles currently available in the Tcl interpreter.

```
datlist
```

datset

This sets the datum attributes to the specified values.

```
datset datId key VALUE ?key2 VALUE2...?
```

Each key must be a valid datum attribute key.

When setting the `TYPE`, the supplied value must be a valid hci data type. No checking or verification is performed on the value.

To destroy only the datum handles created in a particular procedure or section of code, the `hcidatlistreset` command destroys all but a particular set of datum handles. This is located in the system library.

`hcidatlistreset`'s only argument is the list of datum handles to preserve.

This command returns an empty string.

Files commands

The files commands consist of `statfs`, which is used to obtain status information about a file system.

statfs

This obtains status information about a file system.

```
statfs path ?item?|?stat arrayvar?
```

These keys are used to identify data items:

- `version` is the version of status data, currently 0.
- `type` is the type of info, currently 0.
- `bsize` is the fundamental file system block size.
- `blocks` is the total data blocks in the file system.
- `bfree` is the free blocks in the file system.
- `bavail` is the free blocks available to non-root users.
- `files` is the total file nodes in the file system.
- `ffree` is the number of free file nodes.
- `fname` is the file system name. This is the mount point. This is truncated at 6 characters.
- `fpack` is the file system pack name. This is truncated at 6 characters.

If one of these keys is specified as `item`, then that data item is returned.

If `stat arrayvar` is specified, then the information is returned in the array `arrayvar`. Each of the above keys indexes an element of the array containing the data.

If only `path` is specified, then the command returns the data as a keyed list.

GDBM commands

GDBM commands include:

- `gdbm_close`
- `hcigdbmconvert`
- `gdbm_delete`
- `gdbm_fetch`
- `gdbm_firstkey`
- `gdbm_insert`
- `gdbm_list`
- `gdbm_load_package`
- `gdbm_nextkey`
- `gdbm_open`
- `gdbm_reorganize`
- `gdbm_replace`
- `gdbm_store`

`gdbm_close`

This closes a gdbm database with the name *name*.

```
gdbm_close name
```

`hcigdbmconvert`

This promotes a GDBM file from versions before 3.8.1P rev1. This utility converts null-terminated databases to non-terminated, and resides in `$HCIR00T/integrator/bin`.

The GDBM record format was changed beginning with version 3.8.1P rev1 as part of the Tcl internationalization character rework. Previously, both the key and value were put into the database as null-terminated strings, including the null character. With the Tcl changes, the null character is omitted. In the GDBM interface, keys and values are specified as type datum. Datum is a struct that contains a byte array plus the array length.

This adapts the changes in Tcl which support international character sets and binary data. Tcl data are no longer null-terminated strings. They are dual-ported Tcl objects which can have a string representation and an internal representation simultaneously.

Tcl strings are counted, including null-terminated, and can contain embedded nulls. Strings are kept in UTF-8 internally in the interpreter. By default, they are converted from and to the external encoding during input/output operations. An extension is made to the UTF-8 encoding. Embedded null characters are encoded using a 2-byte value that is not part of the normal UTF-8 encoding. In this way, embedded nulls, which are part of the data can be distinguished from the string terminators, which are not part of the data.

The GDBM Tcl extensions were set up to efficiently handle strings, including international characters, and binary data, and not to save 2-bytes per record. Now, the Tcl extension moves the data into the extension as a `Tcl_Obj` type.

-
- For binary data, thisFor string data, it performs a conversion on the data from UTF-8 to the external encoding.

Tcl does not null-terminate ByteArray data. Therefore, there is no null-terminator available. To put data into the database with an extra null-terminator would require making another copy of the data.

When GDBM database null-terminated keys and values are extracted with the Tcl extensions, the trailing null is treated as part of the key or value. When the key/value is moved back into the interpreter, this trailing null is converted to the 2-byte value. For binary data, this returns a pointer to the ByteArray internal representation. `\xc0\x80` during the conversion to UTF-8.

In versions before 3.8.1P, the GDBM databases used null-terminated keys. A supplied utility converts this key and value format, to the new format without terminating null characters.

```
hcigdbmconvert oldDB newDB
```

- `oldDB` is the existing old-format GDBM database.
- `newDB` is the new-format GDBM database that is to be created.

gdbm_delete

This deletes database keys.

```
gdbm_delete name key
```

name is the name of a gdbm database previously opened with `gdbm_open`.

It searches for *key* and deletes it from the database. If *key* is not found, then an error is generated.

This command returns an empty string.

gdbm_fetch

This returns the value that is associated with the key from the gdbm file.

```
gdbm_fetch name key ?retvar | {}?
```

- *name* is the name of a gdbm database previously opened with `gdbm_open`.
- If you do not specify *retvar*, then the value is returned as the result of the command. In this case, if *key* is not found in the list, then an error results.
- If you do specify *retvar* and *key* is in the file, then the value is returned in the variable *retvar* and the command returns 1. If *key* is not in the file, then the command returns 0, and *retvar* is left unchanged.
- If you specify `{}` for *retvar*, then the value is not returned. When this is finished, you can determine if a key is present in a gdbm file without setting a variable as a side effect.

gdbm_firstkey

This returns the first key, in hash order, in the gdbm database.

```
gdbm_firstkey name
```

name is the name of a gdbm database previously opened with `gdbm_open`.

If no keys are present, then this command returns a null string.

gdbm_insert

This inserts data *content* giving it the key *key*.

```
gdbm_insert name key content
```

name is the name of a gdbm database previously opened with `gdbm_open`.

If data with *key* is already in the database, then an error is generated.

This command returns an empty string.

gdbm_list

This generates a list of all keys in the database.

```
gdbm_list name
```

name is the name of a gdbm database previously opened with `gdbm_open`.

gdbm_load_package

The first time this procedure is called within a Tcl interpreter instance, this loads the gdbm package, which loads the gdbm extension library.

```
gdbm_load_package
```

After running, this procedure becomes a no-op and subsequent calls have no effect.

Note: This procedure has no arguments. It is called the first time any `gdbm_*` command is run. To bypass this, invoke this command directly before running any `gdbm_*` commands to explicitly load the gdbm package.

gdbm_nextkey

This returns the next key, in hash order, in the gdbm database.

```
gdbm_nextkey name key
```

name is the name of a gdbm database previously opened with `gdbm_open`.

If no further keys are present or the specified key is not present, then it returns a null string.

gdbm_open

This opens the database with a specified access mode.

```
gdbm_open name ?mode?
```

name opens a gdbm database of that name.

mode can be one of:

- `r` This only reads.
- `w` This only writes.
- `wc` This writes and creates if not already existent.
- `n` This writes and creates a new database, regardless if one exists.

If the *mode* is not given, then it is opened for reading. The gdbm database can be opened many times for reading. After the gdbm database is opened for writing, it is locked, and no other user can read it.

This command returns an empty string.

gdbm_reorganize

This reorganizes the database file. For example, it reduces the size of the database file if there have been many deletions.

```
gdbm_reorganize name
```

name is the name of a gdbm database previously opened with `gdbm_open`.

This command returns an empty string.

gdbm_replace

This replaces the content of a specified key.

```
gdbm_replace name key content
```

- *name* is the name of a gdbm database previously opened with `gdbm_open`.
- If *key* already exists, then its data is replaced with *content*.
- If *key* does not exist, then an error is generated.

This command returns an empty string.

`gdbm_store`

This inserts or replaces data in the database with the key *key*.

```
gdbm_store name key content
```

- *name* is the name of a gdbm database previously opened with `gdbm_open`.
- If *key* already exists, then its data is replaced with *content*.
- If *key* does not exist, then an error is generated.

This command returns an empty string.

Global variables commands

These global variables are available for javaUPoC and JavaDriver:

- `gvsubstring`

This substitutes global variables in a TCL string with their values.

Usage:

```
gvsubstring stringvalue
```

For example, when there is a `secret` global variable that is set to `gofish` in the configuration, the interaction with the command is:

```
hcitcl>gvsubstring {secret is $$secret}  
secret is "gofish"
```

- `gvsubfile`

This loads a file and substitutes all global variable represented in it with their values.

Usage:

```
gvsubfile filename [outfilename]
```

If *outfilename* is given, then the updated result is written to the path pointed by *outfilename*. Otherwise, the updated content is returned by the command as a TCL string.

GRM commands

GRM commands include:

- `grmbulkcopy`
- `grmcreate`
- `grmdestroy`
- `grmencode`
- `grmfetch`
- `grmlist`
- `grmreset`
- `grmstore`

grmbulkcopy

This copies all the record data from `srcGrmId` to `destGrmId`.

```
grmbulkcopy ?-warn var? srcGrmId destGrmId
```

The GRM objects must be the same type (for example, FRL, VRL, XML, HRL, HL7, X12, or EDIFACT). Data are copied from the source to the destination according to the GRM type.

- FRL/VRL/XML/HRL: Only field names common to both the input and output sides are copied. When copying each field, the subfields are matched one-to-one until one or both lists terminate.
- HL7: Input data paths are copied directly to the output side unchanged. After the data are copied, they are available for retrieving from the output side. When the data are encoded into a message, only the relevant paths are used.

When copying between versions, for example, 2.1 and 2.2, a reasonable effort is made to map analogous standard fields. User-defined field IDs are mapped directly. This assumes the same field ID is used in the same segment in both the input and output definitions.

- X12: Input data paths are copied directly to the output side unchanged. After the data are copied, they are available for retrieving from the output side. When the data are encoded into a message, only the relevant paths are used.
- EDIFACT: Input data paths are copied directly to the output side unchanged. After the data are copied, they are available for retrieving from the output side. When the data are encoded into a message, only the relevant paths are used.

If you specify a warning variable name, then *var*, any warning messages generated during the operation are collected in the named variable.

This command returns an empty string.

grmcreate

This creates a new GRM handle for a record format.

```
grmcreate ?-msg msgId? ?-warn var? type args
```

args parsing is based on type:

- FRL

```
frlname
```

frlname is the name of FRL to use

- VRL

```
vrlname
```

vrlname is the name of VRL to use

- HRL

```
hrlname
```

hrlname is the name of HRL to use

- HL7

```
vers variant msgType
```

vers is the version (example, 2.1)

variant is the variant name (example, var1 or {})

msgType is the HL7 message type (example, ADT_A01)

- XML

```
package ocmname ?rootnode?
```

package is the XML package

ocmname is the name of OCM to use

rootnode is the root node of OCMOCM (optional)

- X12

```
vers variant msgType
```

vers is the version (example, 004030)

variant is the variant name (example, var1 or {})

msgType is the X12 message type (example, 835)

- NCPDP

```
vers variant msgType
```

vers is the version (example, NCPDP5.1)

variant is the variant name (example, var1 or {})

msgType is the NCPDP message type (example, P4-RESPONSE)

- NCPDPSCRIPT

```
vers variant msgType
```

vers is the version (example, 4.2)

variant is the variant name (example, var1 or {})

msgType is the NCPDPSCRIPT message type (example, GETMSG)

- NCPDBFAB

```
vers variant msgType
```

vers is the version (example, 1.0)

variant is the variant name (example, var1 or {})

msgType is the NCPDBFAB message type (example, XHD)

- UN/EDIFACT

```
vers variant msgType
```

vers is the version (example, 97B)

variant is the variant name (example, var1 or {})

msgType is the UN/EDIFACT message type (example, AUTHOR)

- HPRIM

```
vers variant msgType
```

vers is the version (example, 2.2)

variant is the variant name (example, var1 or {})

msgType is the HPRM message type (example, ADM)

- LDL

```
vers variant msgType
```

vers is the version (example, 2014)

variant is the variant name (example, var1)

msgType is the LDL message (example, C_MOVE)

- DB

```
dbconnection tableschema ?tableschema ...?
```

dbconnection is the name of database connection

tableschema is the name of tableschema

If you supply a valid msgId, then that message is available to the GRM handle for parsing and data retrieves.

If you specify a warning variable name, then *var*, any warning messages generated during the operation are collected in the named variable.

This command returns a new GRM handle for use in other commands. Use multiple GRM handles to access the same definition in separate messages without interfering with each other.

grmdestroy

This destroys the specified GRM handles.

```
grmdestroy grmId ?...?
```

This command returns an empty string.

grmencode

This encodes the data stored in the handle according to the handle's format.

```
grmencode ?-sepchars keylist? ?-meta metadata? ?-class class?  
?-type type? ?-recover? ?-warn var? grmId
```

The data state of the GRM handle remains unchanged.

If you specify a warning variable name, *var*, then any warning messages generated during the operation are collected in the named variable.

By default, this command returns the newly created message handle of class ENGINE, type DATA, and with the USERDBRECOVER flag off.

grmfetch

This retrieves data from the specified field within the GRM handle.

```
grmfetch ?-warn var? grmId field
```

Depending on the underlying record format, a single field string can imply multiple values. For example, an FRL field that is defined with multiple subfields.

If you specify a warning variable name, *var*, then any warning messages generated during the operation are collected in the named variable.

This command returns a list of datum handles, one per value, such as subfields for FRL or subcomponents for HL7.

An error is returned when the field specification is invalid with respect to the record definition. An error is also returned if any item's value cannot be retrieved. For example, the value is not present in the GRM handle data state, or the underlying message is too short to cover the field.

grmlist

This returns a list of valid GRM handles currently available in the Tcl interpreter.

```
grmlist
```

grmnames

This retrieves field names that are associated with a GRM object.

This function accesses field names from message definitions. For example, HL7, FRL, X12, and so on. The TCL interface provides an interface to return both parsed data and the field name of the data from the definition.

A message can be fully parsed through the TCL interface. The fields named from the message definition are also mapped to each data element.

For example, with the TCL GRM interface you can parse a message with an assigned message definition.

Calling `grmnames` with `grmfetch` returns the associated field names. For example:

```
grmfetch $message definition.frl
grmnames $message variant.hl7
```

`grmfetch` returns the fully parsed message.

`grmnames` returns the field names from the message definition. You can then synchronize the two return objects.

grmnames \$grmId

This command gets the field names list.

This supports FRL, VRL, HRL, HL7, X12, Edifact, HPRIM, LDL, NCPDP, JSON, and XML.

The usage is `grmnames $grmId`, where `grmId` is the GRM object handle.

Example:

```
set grmId [grmcreate -msg $mh frl $fn]
set fieldNames [grmnames $grmId]
puts "${fieldNames}"
foreach field $fieldNames {
  set datums [grmfetch $grmId $field]
  set valueAll ""
  foreach datum $datums
    {set value [datget $datum VALUE]}append valueAll $valuedatdestroy $datum
  }
  grmdestroy $grmId
```

grmreset

This clears the data state.

```
grmreset ?-warn var? grmId ?msgId?
```

If you specify `msgId`, then this command also parses that message.

If you specify a warning variable name, `var`, then any warning messages generated during the operation are collected in the named variable.

This command returns an empty string.

grmstore

This stores one or more values in a GRM handle's data-state according to field.

```
grmstore ?-list? grmId field type args
```

Store both strings and datum objects using this command.

If you use the `-list` option and more values are supplied than are implied by field, then the extra values are ignored. `args` is a list containing all the values to store. Otherwise, the value list is all the remaining arguments to the command.

The `type` value identifies the default item type the list uses. The datum can be present without identification. These values are `c`, `d`, and `v`.

- Constant strings use `-c`.
- Datum objects use `-d`.
- Special values use `-v`: `-c string`, `-d datId`, `-v null`.

The only special value currently supported is `null`. If more values are supplied than are implied by field, then the extra values represent a value that is present, but empty. For example, in HL7 this is encoded as `""` (two double-quotes). It is not treated specially in FRLs.

Constant string values default to type `ch` when stored. You can specify an `hci` data type with a `-type` data type trailer:

```
-c string -type datatype
```

Examples:

```
grmstore grm0 PatientName d datum0 datum1
grmstore -list grm0 {PatientName.[0,1]} d "datum0 datum1"
grmstore grm1 0(0).PID(0).00041 d -c Smith datum2 -v null
```

This command returns an empty string.

Lists commands

Lists commands include:

- `compare`
- `lregex`
- `qlsort`

compare

Compares two lists and returns their relation. For example, `strcmp <0, 0, >0`.

```
compare list1 list2
```

Corresponding elements of each `list` are compared. The result is the value of the first difference encountered or equality. A list that is a prefix of the other list is considered to be less than the other list.

By default, items are compared lexicographically. The `-numeric` flag, or some unique fraction thereof, causes the items to be compared as double-precision floating-point numbers.

lregex

Returns a list of all the elements in `list` that match the regular expression `exp`.

```
lregex list exp
```

qlsort

Sorts the elements of `list`.

```
qlsort list sortproc
```

`sortproc` is the name of a procedure that must accept two arguments.

It must return `<0`, `=0`, or `>0`, indicating the relative ordering of the first and second arguments. For further information, see the manual page for `qsort`.

This command returns a new list ordered according to .

Message commands

Message commands include:

- `adddatamap`

- `msgappend`
- `msgappread`
- `msgcopy`
- `msgcreate`
- `msgdestroy`
- `msgdump`
- `msgerror`
- `findchar`
- `finduchar`
- `msgget`
- `msggetdat`
- `msginsert`
- `msginsread`
- `msglength`
- `msglist`
- `msglock`
- `msgmapdata`
- `msgmetaget`
- `msgmetaset`
- `msgread`
- `msgrouteget`
- `msgrouteset`
- `msgset`
- `msgstats`
- `msgmsgwrite`
- `msgXSLT`

adddatamap

This adds a data mapping table.

```
adddatamap mapName mapList
```

`mapList` is a list of 256 elements: one for each ASCII value. Each element is the ordinal value of the character to which the original character is mapped when you use the table.

After you create a table, a message is mapped using the `msgmapdata` command.

This command returns an empty string.

msgappend

This appends the specified data to `msgId`.

```
msgappend ?-cvtnull char? msgId data
```

If you use the `-cvtnull` option, then every instance of `char` is converted to a null character before the data is appended to the message.

This command returns an empty string.

msgappread

This reads the next message from `fileId`, an open file handle, and appends the data to `msgId`.

```
msgappread ?-stats statsVar? len10 msgId fileId  
msgappread ?-stats statsVar? nl msgId fileId  
msgappread ?-stats statsVar? raw msgId fileId len
```

If you specify a `len10` or `nl` read, then use the appropriate message boundary to terminate the read. Supply a read length, `len`, for raw reads.

If you supply a `statsVar` variable name, then the variable is set to contain a list of two values:

- `todo done`
`todo` is the number of bytes requested. This is the length implied by a `len10` record header, 0 for `nl` reads, or the `len` value for raw reads.
`done` is the actual number of characters read. These values are useful when a read fails.

This command returns the number of bytes read.

msgcopy

This produces a new message with data identical to `msgId`.

```
msgcopy msgId
```

Some metadata fields are copied, some are not. For example, the new message receives a different `mid` and its `srcmid` is set to the original message's `mid`.

This command returns the new message handle.

msgcreate

This creates a new message.

```
msgcreate -meta ?CLASS? ?TYPE? USERDBRECOVER ?data?
```

- `CLASS` specifies an hci message class, engine or protocol.
- `TYPE` specifies an hci message type, data or reply.
- `USERDBRECOVER` specifies that the message undergoes normal recovery database processing when it leaves the interpreter and enters the engine. By default, new messages are `TYPE` data, indeterminate `CLASS`, and not handled by the recovery database.

If specified, then *data* is the initial data value for the new message.

This command returns the new message handle.

msgdestroy

This destroys the specified messages.

```
msgdestroy msgId ?msgId...?
```

msgdump

This prints internal details about the message to stdout.

```
msgdump msgId
```

This command returns an empty string.

To dump message metadata into the log except message data, use this command:

```
msgdump mh -nodumpdata
```

msgerror

This transitions the given messages to the error database and attaches the context string to it.

```
msgerror context msgId ?msgId...?
```

You can put messages into the error database when not in a TPS environment, when you cannot use the `ERROR` disposition. Unlike the `ERROR` disposition, you can attach a context description string to the message.

The command fails if a message is destroy-locked.

This command returns an empty string.

msgfindchar

This finds the first instance of `char` in `msgId`'s region.

```
msgfindchar msgID char ?offset? ?length?
```

- If *offset* is not specified, then 0 is assumed.
- If *length* is not specified or specified as end, then the remainder of the message is used.
- If the character is not in the region, then the command returns -1. If the character is in the message's data, then the command returns the offset from the start of the region.

Use `{}` to search for the null character.

msgfinduchar

This finds the lowest character greater than or equal to `char` not used in the message region.

```
msgfinduchar msgId char ?offset? ?length?
```

- *offset* defaults to 0.
- *length* defaults to the rest of the message, or can be the end.

msgget

This retrieves message data from the region that is specified by `offset` and `length`.

```
msgget ?-cvtnull char? msgId ?offset? ?length?
```

Nulls in the retrieved data are converted to `char` if you use `-cvtnull`. If the message contains nulls and you do not use the `-cvtnull` option, then you cannot access message data past the Tcl message's first null.

In addition, if you store the string returned in this non-conversion case, then only the Tcl-visible string portion is stored.

This command returns the retrieved, converted data.

msggetdat

This returns message data from the region that is specified by `offset` and `length` in a datum object.

```
msggetdat ?-cvtnull char? ?-type type? msgId ?offset? ?length?
```

- Nulls in the retrieved data are converted to `char` if you use `-cvtnull`.
- The datum object is type `ch` unless you specify another *type* value.

This command returns the new datum object's handle.

msginsert

This inserts data into `msgId` at `offset`. The default is zero.

```
msginsert ?-cvtnull char? msgId data ?offset?
```

Instances of `char` in `data` are converted to null if you use `-cvtnull`.

This command returns an empty string.

msginsread

This reads the next message from `fileId`, an open file handle. Then, it inserts the data into `msgId` at *offset* of 0, or *offset*, if specified.

```
msginsread ?-stats statsVar? len10 msgId fileId ?offset?
msginsread ?-stats statsVar? nl msgId fileId ?offset?
msginsread ?-stats statsVar? raw msgId fileId len ?offset?
```

If you specify a `len10` or `nl` read, then use the appropriate message boundary to terminate the read. Supply a read length, `len`, for raw reads.

If you supply a *statsVar* variable name, then the variable is set to contain a list of two values:

```
todo done
```

- `todo` is the number of bytes requested. This is the length implied by a `len10` record header, 0 for `nl` reads, or the `len` value for raw reads.
- `done` is the actual number of characters read. These values are useful if a read fails.

This command returns the number of bytes read.

msglength

This returns the length of `msgId`'s data.

```
msglength msgId
```

msglist

This returns a list of valid message handles currently available in the Tcl interpreter.

```
msglist
```

msglock

This modifies and queries a message object's locks.

```
msglock ?-nodestroy? ?-nowrite? msgId
```

Each flag specifies a lock to set on the message in the interpreter. After set, you cannot remove a lock.

This command returns a list describing the message's lock-state after the new locks, if any, are applied.

The list contains zero or more of these elements:

- `nodestroy`: Message cannot be destroyed.
- `nowrite`: Message cannot be modified.

msgmapdata

This applies the character conversion described by `maptable` to *msgId*.

```
msgmapdata msgId maptable
```

Create map tables using the `adddatamap` command.

This command returns an empty string.

msgmetaget

This returns metadata keys.

```
msgmetaget ?-rw | -ro | -all? msgId ?key?
```

- `-rw` restricts the key list to read-write (modifiable).
- `-ro` restricts the key list to read-only.
- If you do not specify a *key* value, then the available metadata keys are returned.
- If you specify a *key* value, then the metadata that is associated with that key is returned.

msgmetaset

This sets values for one or more metadata keys.

```
msgmetaset msgId key value ?key2 value2 ...?
```

This command returns an empty string.

msgread

This reads the next message from *fileId*, an open file handle, and replaces some part of *msgId* with the data read.

```
msgread ?-stats statsVar? len10 msgId fileId ?off? ?len?
msgread ?-stats statsVar? nl msgId fileId ?off? ?len?
msgread ?-stats statsVar? raw msgId fileId len ?off? ?len?
```

The replacement region begins at offset *off*, or 0, if not specified. It includes *len* bytes, or the remainder of the message data, if not specified. If you specify a *len10* or *nl* read, then use the appropriate message boundary to terminate the read.

Supply a read length, *len*, for raw reads.

If you supply a *statsVar* variable name, then the variable is set to contain a list of two values:

```
todo done
```

- *todo* is the number of bytes that are requested. This is the length implied by a *len10* record header; 0 for *nl* reads, or the *len* value for raw reads.
- *done* is the actual number of characters read. These values are useful when a read fails.

This command returns the number of bytes read.

msgrouteget

This returns *msgId*'s translation and routing configuration list.

```
msgrouteget msgId
```

You can only use this command in generate route procedures.

msgrouteset

This replaces *msgId*'s translation and routing configuration list with *list*.

```
msgrouteset msgId list
```

You can only use this command in generate route procedures. *list* is a list of keyed lists, each of which describes a translation route detail.

msgset

This replaces a portion of `msgId`'s data with data.

```
msgset ?-cvtnull char? msgId data ?offset? ?length?
```

Instances of `char` are converted to null if you use `-cvtnull`. The replacement region begins at `offset`, or 0, if not specified. It includes `length` bytes, or the remainder of the message, if not specified.

msgstats

This returns a keyed list containing statistics on message memory use.

```
msgstats msgId
```

msgwrite

This writes data from *msgId* to file handle *fileId* using some record style.

```
msgwrite style msgId fileId ?offset? ?length?
```

The region that is written begins at *offset*, or 0 if not specified, and includes *length* bytes. This is the remainder of the message if not specified.

This command returns an empty string.

msgXSLT

This transforms an XSLT on an XML message handle in Tcl. The input message is transformed using the given XSLT file. It then returns the output in a new message handle. The new handle inherits the metadata from the input message handle.

```
msgXSLT msgId xsltfn ?key VALUE? ?key2 VALUE2 ...?
```

- `msgId` is the message handle of the input data, and must contain a valid XML message.
- `xsltfn` is the file name of the XSLT file.
- `key VALUE` is the set of runtime parameters to pass to the XSLT engine.

MSI commands

MSI commands include:

- `msiAttach`
- `msiGetStatSample`
- `msiTocEntry`

msiAttach

This attaches the MSI memory region.

```
msiAttach
```

The first time this command is run in an interpreter, it returns an empty string. An error happens every subsequent time.

Note: You must use this command before the other `msi` extensions work.

msiGetStatSample

This works such as `keylget` for `retvar` handling.

```
msiGetStatSample {threadName | index} ?retvar?
```

You can specify the thread from which to extract data by name or by index.

- If the argument is wholly numeric, then it is assumed to be an index. If not, then it is assumed to be a thread name.
- If you do not specify `retvar`, then the value returned is the keyed list of static data. If the thread is not found within the ToC, then an error results.
- If you specify `retvar` and the thread is in the ToC, then the keyed list of data is placed in `retvar` and the command returns 1. If the thread is not in the ToC, then the command returns 0 and `retvar` is left unchanged.
- If the specified thread exists but has never been initialized, then an empty keyed list results.

msiTocEntry

This works such as `keylget` for `retvar` handling.

```
msiTocEntry ?threadName? ?retvar | {}?
```

- If you do not specify `retvar`, then the value is the keyed list of ToC entry data. If the thread is not found within the ToC, then an error results.
- If you specify `retvar` and the thread is in the ToC, then the keyed list of data is placed in `retvar` and the command returns "1". If the thread is not in the ToC, then the command returns 0 and `retvar` is left unchanged.

- If you specify *retvar* as "{}", then the keyed list is not returned, permitting the programmer to determine the presence of the ToC entry.
- If you omit *threadName*, then the list of threads in the ToC is returned.

Strings commands

The `strsub` command replaces the characters from `first` to `last` in `string1` with the contents of `string2`.

```
strsub string1 first last ?string2?
```

- If `string2` is not present, then an empty string is used. This effectively deletes the specified characters. Both `first` and `last` can be specified as `end`, or any abbreviation of it, to refer to the last character in `string2`.
- If `first` is less than zero, then it is treated as if it were zero. If `last` is greater than or equal to the length of the string, then it is treated as if it were `end`. It is an error for `first` to be greater than `last`.

This command returns the modified string.

Table commands

Table commands include:

- `tbllookup`
- `dblookup`
- `dblookupdestroy`

tbllookup

This maps a value through a table.

```
tbllookup ?-side side? table value
```

`side` is user-defined and only available if the table is defined as bi-directional.

The default sides are input and output.

If the `value` is not found, then this command returns nothing.

Example:

```
set groupname [tbllookup groups 26]  
set docid [tbllookup -side output docs "Dr. Watson"]
```

dblookup

```
dblookup ?-maxrow ?count?? ?-metacolumnname? table value ?value...?
```

This looks up database data through a database lookup table that is based on the given value. The table is a database lookup file name. The file extension can be omitted.

The input values provided by `dblookup` are set as the IN column values one-by-one. The input value account equals the IN column account that is defined in the `.tbl` file. If they are not matched, then an error is prompted and the Database Lookup terminated.

- `-maxrow ?count?`
This sets the maximum returned row count. By default, at most one record is returned from the `dblookup` command. When you use this option, at most the specific *count* records are returned. To get all the selected records, set this option without *count*.
- `-metacolumnname`
This gets the column names from result set metadata. With this option, both the selected database data and column names of result set metadata are returned in a keyed list.

The `dblookup` Tcl command can be used in engine Tcl UPoCs for the translation thread and in the inbound/outbound protocol thread UPoCs.

dblookupdestroy

This destroys the related resources, for example, JNI context and database connection, used by `dblookup`. This command returns an empty string.

Example:

```
dblookupdestroy table
```

`dblookupdestroy` is not required when using `dblookup` at engine runtime. The engine automatically destroys the resources that are used by the Database Lookup table during shutdown.

Examples

```
hcitcl>dblookup test_simple.tbl key01
test1

hcitcl>dblookup test.tbl key01 test1
1,2014-04-01,0,test1,key01

hcitcl>dblookup -metacolumnname test.tbl key01 test1
{RSMETACOLUMNNAME {intFLD} {dateFLD} {flag} {strFLD} {ID}} {RSDATA {{1} {2014-04-01} {0} {test1}
{key01}}}}

hcitcl>dblookup -maxrow 2 test_rows.tbl 0
2014-04-01,0,key01,1,test1
2014-04-02,0,key02,2,test2

hcitcl>dblookup -maxrow 2 -metacolumnname test_rows.tbl 0
{RSMETACOLUMNNAME {dateFLD} {flag} {ID} {intFLD} {strFLD}} {RSDATA {{2014-04-01} {0} {key01} {1}
{test1}} {{2014-04-02} {0} {key02} {2} {test2}}}}
```

```

hcitcl>dblookup -maxrow test_rows.tbl 0
2014-04-01,0,key01,1,test1
2014-04-02,0,key02,2,test2
2014-04-03,0,key03,3,test3
2014-04-04,0,key04,4,test4
2014-04-05,0,key05,5,test5
2014-04-06,0,key06,6,test6
2014-04-07,0,key07,7,test7
2014-04-08,0,key08,8,test8
2014-04-09,0,key09,9,test9
2014-04-10,0,key10,10,test10
hcitcl>dblookupdestroy test_rows.tbl
hcitcl>

hcitcl>dblookupdestroy test
hcitcl>

```

Xlate commands

The `convert_date` command converts between the hci supported date and time formats.

```

convert_date [-configure keyedlist] inType inData outType\ outLen
convert_date [-configure keyedlist]
convert_date -configure

```

- `inType` and `outType` entries must be one of the supported type codes.
- `outLen` determines which of any optional components are included in the output data.

This command returns the converted string.

Note: Using `convert_date` without any parameters returns the current GD configuration. In this way, you can display the options that were set using `-configure`, make any temporary changes, and then change the settings back.

The supported date and time codes are:

- `dt`: [CC]YYMMDD
- `ed`: DDMM[CC]YY
- `fd`: MM/DD/[CC]YY
- `fe`: DD.MM.[CC]YY
- `jd`: [[CC]YY]DDD (Julian)
- `tm`: HHMM[SS][{+|-}ZZZZ]
- `ts`: YYYYMMDDHHMM[SS][{+|-}ZZZZ]
- `yd`: MMDD[CC]YY

Use any separator characters for `fd` and `fe` input; the characters shown are used when those types are specified as `outType`.

The type codes are case insensitive. Any conversions that are incomplete use the current date and time as the source for any missing information. "Incomplete" indicates the output format has more information than the input.

For example, converting a `dt` with a value of 19930410 to a type `ts` with a length of 12, results in 199304101415. This assumes the current time is 2:15 PM.

Time zones, when requested in the output format, are normalized to the local time zone. This is in accordance with any time zone shift that is specified in the input data.

If `inData` is empty, then `inType` is ignored and the output is formatted using only data elements corresponding to the current date and time.

Example commands

```
convert_date -configure [list {FABRICATE 0}\
{RANGE {50 20 19}}]
```

```
convert_date -configure [list {ADDPREC 0}\
{DELIMIT {/ :}} {FABRICATE 1}]
```

Setting ADDPREC

The `ADDPREC` key affects not only the inclusion of the precision for `convert_date`. It also affects whether the current date and time are used to complete missing format elements.

To get the current date and time, `ADDPREC` must be set properly.

For `DATECOPYOPT`, the default is no precision, which affects the functioning of `convert_date` inside and outside of `xlates`. For example, TPS procs.

For example, in:

```
convert_date ch "" ts 12
```

`convert_date` returns 2010, which is only the current year.

If you add the `ADDPREC` key, then:

```
convert_date -configure [list {ADDPREC 1}]
```

and then use:

```
convert_date ch "" ts 12
```

`convert_date` returns 201001051125, which is the full date and time.

You can also use:

```
convert_date -configure [list {ADDPREC 1}] ch "" ts 12
```

With `ADDPREC` enabled, if you use:

```
convert_date ch "" ts 19
```

`convert_date` returns 20100105112800-0600, the offset for Central time.

Keyed list option

The scope of the `-configure` option can change based on your configuration, such as multi-threaded translations. You should set the `-configure` options at the top of each procedure that uses `convert_date` and which expects certain options to be set.

The `-configure` keyed list option permits a keyed list of date conversion options to be passed in and permanently set. These keys and options are the same ones used in the `DATECOPYOPT` xlate operation.

This table lists the options that control the behavior of the date conversion operation:

Key	Value	Description
FABRICATE	boolean	<p>This is used in tandem with the <code>RANGE</code> key.</p> <ul style="list-style-type: none"> If set to 1, then the default century that is added to the output string is the current century. This is only if the century is not present on the input side. If set to 0 (false), then the <code>RANGE</code> key must be present to tell the conversion operation how to calculate the century. This applies only if a century needs to be added to the output. <p>For example:</p> <pre>{FABRICATE 1}</pre>
RANGE	Three integer list	<p>The first value in the <code>RANGE</code> key's list is the border, the next is the low century and the third is the high century.</p> <ul style="list-style-type: none"> If the year in the input string is less than or equal to the border, then the output uses the low century. If the year in the input string is greater than the border, then the output uses the high century. No default and <code>FABRICATE</code> is false. <p>For example:</p> <pre>{RANGE {25 20 19}}</pre>
DELIMIT	List of two delimiter characters	<p>The value of the <code>DELIMIT</code> key is a list that specifies the two characters to use as the date separator character and time separator character. The default separator for dates is the <code>/</code> character, except for type <code>fe</code> which defaults to <code>"."</code>. The default for time is the <code>":"</code> character.</p> <p>For example:</p> <pre>{DELIMIT {- .}}</pre>

Key	Value	Description
ADDPREC	boolean	<p>The ADDPREC key's value tells the date conversion operation whether to add precision to the output date/time string.</p> <ul style="list-style-type: none"> If the value is true, or 1, then fields such as seconds or timezone are added to the output string, created from the current date/time. If the value is false, or 0, then optional fields not present in the input are not added to the output string. For example, if a time is 145510 and this flag is false, no fractions of a second or timezone are added to the output. <p>For example:</p> <pre>{ADDPREC 0}</pre>
USECURTM	boolean	<p>Setting this value means the command references the system time when filling in missing values in the incoming date/time value.</p> <ul style="list-style-type: none"> When set to 1, <code>convert_date</code> uses the system time. When set to 0, <code>convert_date</code> uses the time values defined in TMD EFS. <p>For example:</p> <pre>{USECURTM 1}</pre>
TMDEFS	Tcl list	<p>When USECURTM is set to 0, this key defines which values to use when filling in missing values in the incoming date/time value.</p> <p>This splits the list of defaults for the various date and time related fields. All fields are optional and can be an empty list.</p> <p><code>now</code> can be used in any field to represent the value is filled in from the current time.</p> <p>For example: Midnight January 1 would be:</p> <pre>{TMDEFS {01 01 00 00 00 0000 +0000}}</pre>

XPM commands

XPM commands include:

- `xpmdispose`
- `xpmencode`
- `xperror`
- `xpmfetch`
- `xpmmetaget`

- `xpmmetasest`
- `xpmstore`

xpmdispose

This places one or more messages onto `xpmId`'s output list with disposition `disp`.

```
xpmdispose xpmId disp msgId ?...?
```

`disp` must be one of these TPS disposition values:

- `continue`
- `kill`
- `over`
- `proto`
- `send`

Messages that are passed to XPM by this command are no longer available in the interpreter.

This command returns an empty string.

xpmencode

This encodes a message according to the XPM output data state, and returns its handle.

```
xpmencode ?-warn var? xpmId
```

Any warnings that are generated during the encoding process are appended to the variable `var`.

Encoding a message does not affect the translation process. Messages that are produced by this command exist only in the interpreter. You are responsible for destroying them, or passing them to the engine by `xpmdispose`.

xpmerror

This generates an XPM error condition.

```
xpmerror xpmId severity message
```

`severity` is any of these values:

- `action`: This error is handled by XLT statement error control value.
- `curdetail`: Error out only the current translation routing detail, but continue the remaining ones.
- `alldetail`: Error out all of the message's translation routing details.

`message` is a textual description of the error. If the source message is moved to the error database, then `message` is attached to the record as its error context.

This command also generates a Tcl error event. Handle the Tcl error with the `catch` command. Doing so does not affect the XPM error condition.

xpmfetch

This retrieves data from the specified field within the XPM handle.

```
xpmfetch ?-warn var? xpmId field
```

Depending on the underlying record format, a single field string can imply multiple values. For example, an FRL field that is defined with multiple subfields.

This command returns a list of datum handles, one per value (subfields for FRL, subcomponents for HL7). If any warning messages are generated during the retrieval, then they are appended to `var`, if specified.

An error is returned if the field specification is invalid, with respect to the record definition. An error is also returned if any item's value cannot be retrieved. For example, not present in XPM handle data-state, or the underlying message is too short to cover the `field`.

`field` is interpreted as an input value to an XPM operation. Its leading character identifies which side of the translation the data is retrieved from:

- `@name`: Temporary field name
- `~name`: Output field name
- `name`: Input field name

xpmmetaget

This queries the default metadata from messages that are produced by `xpmId`.

```
xpmmetaget ?-rw | -ro | -all? xpmId ?key?
```

`-rw` and `-ro` restrict the list to the read-write, or modifiable, and read-only key lists.

If you do not specify a `key` value, then the available metadata keys are returned.

If you specify a `key` value, then the metadata that is associated with that key returns.

Manipulating data

You can manipulate metadata in the Translation tool, so that an `xlate` can configure HTTP headers/URL and modify `DRIVERCTL` during `xlate`.

This, and other metadata fields, is accessed in `xlate` using the `xpmmetaget` and `xpmmetaset` commands.

Example:

```
set udata [xpmmetaget $xlateId USERDATA]
```

xpmmetaset

This sets values for one or more metadata keys.

```
xpmmetaset xpmId key VALUE ?key2 VALUE2 ...?
```

This command modifies the default metadata that is inherited by messages produced by `xpmId`.

This command returns an empty string.

Manipulating data

You can manipulate metadata in the Translation tool, so that an `xlate` can configure HTTP headers/URL and modify `DRIVERCTL` during `xlate`.

This, and other metadata fields, is accessed in `xlate` using the `xpmmetaget` and `xpmmetaset` commands.

Example:

```
set udata [xpmmetaget $xlateId USERDATA]
```

xpmstore

This stores values into the output fields of `xpmId`. `field` is interpreted as an output field.

```
xpmstore ?-list? xpmId field TYPE args
```

A leading `@` identifies a temporary field variable. You cannot modify input fields.

If you use `-list`, then `args` is a list containing all of the values to store. Otherwise, the value list is all the remaining arguments to the command.

Both strings and datum objects can be stored using this command.

- Constant strings use `-c`.
- Datum objects use `-d`.
- Special values use `-v`.

For example:

```
-c string -d datId -v value...
```

The type value, `-c`, `-d`, or `-v`, identifies the default item type for the list. The datum can display without identification.

The only special value currently supported is null, which represents a value that is present, but empty. In HL7, for example, it is encoded as "" (two double-quotes); it is not treated specially in FRLs.

Constant string values default to type `ch` when stored. You can specify an `hci` data type with a `-type` data type trailer.

For example:

```
-c string -type datatype
```

Command examples:

```
xpmstore xpm0 PatientName d datum0 datum1
xpmstore -list xpm0 {PatientName.[0,1]} d "datum0 datum1"
xpmstore xpm1 0(0).PID(0).00041 d -c Smith datum2 -v null
```

This command returns an empty string.

HTTP commands

HTTP commands include:

- `httppost`
- `httpget`
- `httpget`

httppost

This posts data to a specified resource, usually a CGI script or form. This command returns a keyed list.

```
httppost { {URL url} {DATA data} ?{HEADERS headerList}? ?{AUTH authList}?
?{HTTPS HTTPSList}? ?{TIMEOUT seconds}? ?{PROXY ?{USER username}? ?{PASS password}?
?{HOST proxyServerMachine}? ?{PORT proxyServerPort}? }? {DEBUG 1}}
```

- `url` is the URL/web address. For example, `www.foo.com` or `http://foo.com:80/index.html`.
- `data` is the data that is sent to the remote resource.
- `headerList` is a keyed list of HTTP header settings. For example:

```
{ {User-Agent Tcl/Tk} {Content-Type text/xml} }
```

- `authList` is a keyed list of user name/password authorization settings:

```
{ {SCHEME scheme} {USER user} {PASSWD password} }
```

- `scheme` is the encryption scheme used to encode the password. For example, `basic`, or `base-64`.
- `user` is the user name.

- `password` is the encrypted password.
- `HTTPSList` is a keyed list of HTTPS security settings to use HTTPS:

```
{CA_FILE cafile} {CA_PATH capath} {CERT_FILE certfile} {PRIVATE_KEY privatekey}
{MODE mode}
```

- `cafile` is the certificate authority file.
- `capath` is the certificate authority path.
- `certfile` is the certificate file.
- `privatekey` is the private key file.
- `mode` is the mode of HTTPS authentication.

Modes are:

- `None`
- `Anonymous`
- `ClientAuth`
- `ClientServerAuth`
- `seconds` is the expiration time in seconds. If the timer expires, `httppost` returns this code:

```
{STATUS {HTTP/1.1 408 Request Time-out}} {HEADERS{}} {BODY{}}
```

- `PROXY` is a container key that is a list of proxy configuration options:
 - `username` is the user name that the connection uses to authenticate the user to the proxy server. If the proxy server is not authenticating, this field is still sent, but is ignored by the proxy server.
 - `password` is the password associated with the user name. If the proxy server is not authenticating, this field is still sent, but is ignored by the proxy server.
 - `proxyServerMachine` is the host name of the computer on which the proxy server is running.
 - `proxyServerPort` is the IP port on which the proxy server listens for connections.
- `DEBUG 1` shows you what is being posted, including the HTTP headers.

`httppost` returns a keyed list containing these keys:

- `STATUS`
A keyed list indicating status. For example, {Version HTTP/1.1} {Code 200} {Desc OK}.
- `HEADERS`
A keyed list of headers. For example, {Set-Cookie name=value} {Server NCSA/1.3}.
- `BODY`
A string containing requested document data.

httpget

This retrieves a specified document from the HTTP server. This command returns a keyed list.

```
httpget {{URL url}} ?{HEADERS headerList}? ?{AUTH authList}? ?{HTTPS HTTPSList}?
?{TIMEOUT seconds}? ?{PROXY ?{USER username}? ?{PASS password}?
?{HOST proxyServerMachine}? ?{PORT proxyServerPort}? }? {DEBUG 1}}
```

- `url` is the URL/web address. For example, `www.foo.com` or `http://foo.com:80/index.html`.
- `headerList` is a keyed list of HTTP header settings. For example:

```
{User-Agent Tcl/Tk} {Content-Type text/xml}}
```

- `authList` is a keyed list of user name/password authorization settings:

```
{{SCHEME scheme} {USER user} {PASSWD password}}
```

- `scheme` is the encryption scheme used to encode the password. For example, basic, or base-64.
- `user` is the user name.
- `password` is the encrypted password.
- `HTTPSList` is a keyed list of HTTPS security settings to use HTTPS:

```
{CA_FILE cafile} {CA_PATH capath} {CERT_FILE certfile} {PRIVATE_KEY privatekey}  
{MODE mode}
```

- `cafile` is the certificate authority file.
- `capath` is the certificate authority path.
- `certfile` is the certificate file.
- `privatekey` is the private key file.
- `mode` is the mode of HTTPS authentication.

Modes are:

- None
- Anonymous
- ClientAuth
- ClientServerAuth
- `seconds` is the expiration time in seconds. If the timer expires, `httpget` returns this code:

```
{STATUS {HTTP/1.1 408 Request Time-out}} {HEADERS{}} {BODY{}}
```

- `PROXY` is a container key that is a list of proxy configuration options:
 - `username` is the user name that the connection uses to authenticate the user to the proxy server. If the proxy server is not authenticating, this field is still sent, but is ignored by the proxy server.
 - `password` is the password associated with the user name. If the proxy server is not authenticating, this field is still sent, but is ignored by the proxy server.
 - `proxyServerMachine` is the host name of the computer on which the proxy server is running.
 - `proxyServerPort` is the IP port on which the proxy server listens for connections.
- `DEBUG 1` shows you what is being posted, including the HTTP headers.

`httpget` returns a keyed list containing these keys:

- `STATUS`
A keyed list indicating status. For example, `{Version HTTP/1.1} {Code 200} {Desc OK}`.
- `HEADERS`
A keyed list of headers. For example, `{Set-Cookie name=value} {Server NCSA/1.3}`.

- **BODY**
A string containing requested document data.

httpput

This publishes data to a remote web server at a specified URL. This command returns a keyed list.

```
httpput {{URL url}} {DATA data} ?{HEADERS headerList}? ?{AUTH authList}?
?{HTTPS HTTPSList}? ?{TIMEOUT seconds}? ?{PROXY ?{USER username}? ?{PASS password}?
?{HOST proxyServerMachine}? ?{PORT proxyServerPort}? }? {DEBUG 1}}
```

- `url` is the URL/web address. For example, `www.foo.com` or `http://foo.com:80/index.html`.
- `data` is the data that is sent to the remote resource.
- `headerList` is a keyed list of HTTP header settings. For example:

```
{{User-Agent Tcl/Tk}} {Content-Type text/xml}}
```

- `authList` is a keyed list of user name/password authorization settings:

```
{{SCHEME scheme}} {USER user} {PASSWD password}}
```

- `scheme` is the encryption scheme used to encode the password. For example, `basic`, or `base-64`.
- `user` is the user name.
- `password` is the encrypted password.
- `HTTPSList` is a keyed list of HTTPS security settings to use HTTPS:

```
{{CA_FILE cafile}} {CA_PATH capath} {CERT_FILE certfile} {PRIVATE_KEY privatekey} {MODE mode}
```

- `cafile` is the certificate authority file.
- `capath` is the certificate authority path.
- `certfile` is the certificate file.
- `privatekey` is the private key file.
- `mode` is the mode of HTTPS authentication.

Modes are:

- `None`
- `Anonymous`
- `ClientAuth`
- `ClientServerAuth`
- `seconds` is the expiration time in seconds. If the timer expires, `httpput` returns this code:

```
{STATUS {HTTP/1.1 408 Request Time-out}} {HEADERS{}} {BODY{}}
```

- `PROXY` is a container key that is a list of proxy configuration options:
 - `username` is the user name that the connection uses to authenticate the user to the proxy server. If the proxy server is not authenticating, this field is still sent, but is ignored by the proxy server.

- `password` is the password associated with the user name. If the proxy server is not authenticating, this field is still sent, but is ignored by the proxy server.
- `proxyServerMachine` is the host name of the computer on which the proxy server is running.
- `proxyServerPort` is the IP port on which the proxy server listens for connections.
- `DEBUG 1` shows you what is being posted, including the HTTP headers.

This command returns a keyed list containing these keys:

- `STATUS`
A keyed list indicating status. For example, {Version HTTP/1.1} {Code 200} {Desc OK}.
- `HEADERS`
A keyed list of headers. For example, {Set-Cookie name=value} {Server NCSA/1.3}.
- `BODY`
A string containing requested document data.

Standard HTTP Tcl procs

These standard Tcl procedures are provided:

- `httpFilesetFetch`
- `httpQuery`

These show how to use the HTTP Client driver. They are also provided as fully-functional procedures for use within system configurations.

You can also create your own customized HTTP procedures according to your site requirements.

FileSet Mode: `httpFilesetFetch` (and `httpDirParse`)

The primary purpose of HTTP Client support is to provide a way to move remote files through firewalls. Typically, a firewall excludes all traffic in and from a server, except for port 80. All port 80 traffic is handled by a web server, thereby requiring all conversations to use the HTTP protocol. The system can connect to remote hosts on port 80 by existing FTP or HTTP protocols. FTP cannot be used to communicate in a system, as is required by the web server. The solution is HTTP.

HTTP Client is modeled after the existing FileSet FTP drivers, functioning in the same basic way. The majority of the differences are protocol level. FileSet FTP sends and receives messages remotely by transferring them as files, either several messages within a single file or a file for each message. HTTP functions similarly using `GET` or `POST` commands to retrieve documents and `PUT` command to send documents.

An outbound transfer is straightforward: the files are sent as files using the FTP protocol. Inbound is more complicated. A directory is given, and some subset of the contained files is retrieved by the FTP protocol. To retrieve the files, the driver must first read the contents of the directory and form a list of available files. This is where the similarity between FTP and HTTP ends. HTTP has no protocol-level directive for listing the directory contents. The most likely solution is to request the directory path as its URL, followed by a slash to indicate that this is a directory. HTTP servers typically respond to such requests with a page listing the files

and subdirectories contained in the directory. This result is returned in formatted HTML. An HTML parser is required to properly turn this HTML content into a concise directory list that is usable by a system driver.

This example shows a typical HTTP server response to a directory contents request:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final //EN">
<HTML>
  <HEAD>
    <TITLE>Index of /Movies/backup/dir1</TITLE>
  <HEAD>
  <BODY>
    <H1>Index of /Movies/backup/dir1</H1>
    <PRE><IMG SRC="/icons/blank.gif" ALT="      " > <A HREF="?N=D">Name</A>
    <A HREF="?M=A">Last modified</A>      <A HREF="?S=A">Size</A><A HREF="?D=A">Description</A>
    <HR>
    <IMG SRC="/icons/folder.gif" ALT="[DIR]" > <A HREF="/Movies/backup/">Parent Directory</A>      26-
    Feb-2010 14:44
    <IMG SRC="/icons/text.gif" ALT="[TXT]" > <A HRF="file1.html">file1.html</A>      26-Feb-2010 14:44
    Ok
    <IMG SRC="/icons/text.gif" ALT="[TXT]" > <A HRF="file2.html">file2.html</A>      26-Feb-2010 14:45
    Ok
    <IMG SRC="/icons/text.gif" ALT="[TXT]" > <A HRF="file3.html">file3.html</A>      26-Feb-2010 14:45
    Ok
    </PRE><HR>
    <ADDRESS>Apache/1.3.12 Server at servername.domainname.com Port 80/ <ADDRESS></BODY></HTML>
```

In this example, the requested directory contains these files:

- file1.html
- file2.html
- file3.html

Parsing out the file names requires a pseudo-parse of the HTML by scanning the text for A HREF tags. It extracts from those tags the enclosed quoted strings. Exceptions are made for quoted strings starting with a question mark (?). This ensures that references such as ?N=D", which is a hyperlink to reorder the list by name, are not included. Another exception is made to omit the parent directory (A HREF="/Movies/backup/").

After a directory list is parsed from the contents page, the list is passed to the user. It is then acted upon by custom Tcl procedures, as in FileSet FTP. It is the responsibility of the user to differentiate files from subdirectories. This causes the user to be aware of subdirectories, in case this information is of use.

Not all web servers present these directory listings in a reasonably consistent way. Among Apache servers, this is fairly consistent, but other web servers occasionally generate slight variations. Therefore, the directory-list parsing task is accomplished through a standard Tcl procedure: `httpDirParse`. In this way, the procedure can be slightly modified as required to accommodate variations of directory-listing formats.

Message-driven mode: httpQuery

The system can also be used to help web-enable legacy applications. This includes the requirement to send message-driven web-queries to remote web servers, and to receive web responses back in the form of messages. These types of interactions are message-driven. Otherwise, a protocol thread is limited to communicating with a single web page.

Other protocol drivers connect to a single remote resource, that is, remote TCP host on a predefined socket port. This provides a constant flow of information. Connecting to a single URL provides only one page of data,

thus becoming a stale source of data. Therefore, the driver can connect to various URLs dynamically, as instructed by messages routed from other system threads.

This is performed by passing HTTP-specific information within the message's metadata. Although it could be passed within the message data itself using metadata, driver data is kept isolated from the message data.

httpQuery and httpFilesetFetch differences

The fundamental difference between `httpFilesetFetch` and `httpQuery` is TPS mode. Because `httpQuery` is message-driven, it needs to run in message-driven mode. This disables the Query Interval feature. Doing this treats the TPS as an outbound TPS that is only called when messages arrive on the outbound queue. The TPS is usually called in `Run` mode, which is important to know if modifications are made to the procedure.

Alternatively, the `httpFilesetFetch` procedure is time-driven, so it needs to run in a timer mode. This enables the query interval feature. The TPS is called on a timed basis according to the specified time interval. This TPS is usually called in `Time` mode.

Example

```
#####
# Name:          httpQuery
# Purpose:       Basic Message-driven HTTP query
# UPoC type:    tps
# Args:         tps keyedlist containing these keys:
#               MODE    run mode ("start", "run" or "test")
#               MSGID    message handle
#               ARGS     user-supplied arguments:
#####
proc httpQuery { args } {
    global cfgs
    keylget args MODE mode
    set dispList {}
    switch -exact -- $mode {
        start
        {
            echo "I AM IN HTTPQUERY START\r"

            # grab and store CFGS values for run mode --
            # CFGS are only available in start mode.
            keylget args CFGS cfgs
        }

        run
        {
            echo "I AM IN HTTPQUERY RUN\r"

            # fetch and process msgid
            keylget args MSGID inMsg

            # fetch the URL using the specified method
            keylget cfgs METHOD method
            switch -exact -- $method {
                PUT {
                    echo "HTTPPUT\r"
                    set res [httpput $cfgs]
                }
                POST {
                    echo "HTTPPOST\r"
                    set res [httppost $cfgs]
                }
            }
        }
    }
}
```

```

    }
    default {
        echo "DEFAULT (GET) \r"
        set res [httpget $cfs]
    }
}

# parse out the response
set status [keylget res "STATUS"]
set statCode [lindex $status 1]
set body [keylget res "BODY"]

if {$statCode == 200} {
    # if OK, place msg into IB queue
    set outMsg [msgcreate -type data $body]
    lappend dispList "KILL $inMsg"
    lappend dispList "OVER $outMsg"
} else {
    # otherwise, report the error and send orig msg to error DB
    echo "ERROR: Failed to fetch resource '$url'."
    echo "Fetch returned:\r$status\r"
    error $inMsg
}
}
time
{
    echo "I AM IN HTTPQUERY TIME\r"
    # Timer-based processing
}

default {
}
}

return $dispList
}
#####
# Name:      httpFilesetFetch
# Purpose:   Fileset-style directory contents fetch for HTTP
# UPoC type: tps
# Args:      tps keyedlist containing these keys:
#             MODE    run mode ("start", "run" or "test")
#             MSGID    message handle
#             ARGS     user-supplied arguments:
#
#####
proc httpFilesetFetch { args } {
    global srcDir
    global cfs
    keylget args MODE mode
    keylget args CFGS cfs
    keylget args ARGS userargs
    keylget userargs F00 foo
    set dispList {}
    switch -exact -- $mode {
        start
        {
            # Perform special init functions
            echo "I AM IN FILESETFETCH START\r"
            # grab and store CFGS values for run mode --
            # CFGS are only available in start mode.
        }
        run
        {
            # Not used...
            # httpFilesetFetch is timer-based, NOT message-driven.
            echo "I AM IN FILESETFETCH RUN\r"
        }
        time
        {
            echo "I AM IN FILESETFETCH TIME\r"
            # Get a list of fetchable URLs contained in the directory
            set urlList [httpDirParse $cfs]
            # Fetch each URL in the list

```

```

        foreach url $urlList {
            set new [list "URL" $url]
            set res [httpget [linsert $cfgs 0 $new ]]
            set status [keylget res "STATUS"]
            set statCode [lindex $status 1]
            set body [keylget res "BODY"]
            if {$statCode == 200} {
                lappend dispList "OVER [msgcreate -type data $body]"
            } else {
                error "ERROR: Failed to fetch resource '$url'.\rFetch re
turned:\r$status\r"
            }
        }
    }
    default {
    }
    return $dispList
}

#####
# Function for parsing a standard Apache-style web-server-
# generated HTML directory listing. The function takes
# the same arguments as the httpget function, passing them
# as-is to httpget. It fetches the HTML directory index
# at the specified URL and returns a list of URLs that can
# then be fetched individually.
#####
proc httpDirParse { args } {
    set arg0 [lindex $args 0]
    keylget arg0 URL url
    set resp [httpget $arg0 ]
    set dir1 [keylget resp "BODY"]
    set dir1 [string toupper $dir1]
    set p1 [string first "<A HREF" $dir1]
    while {$p1 >= 0} {
        set dir1 [string range $dir1 $p1 end]
        set p2 [string first "/>" $dir1]
        set href [string range $dir1 0 $p2]
        set dir1 [string range $dir1 $p2 end]
        set q1 [string first "\"" $href]
        set q2 [string last "\"" $href]
        set ref [string range $href $q1 $q2]
        set ref [string trim $ref "\""]
        if { [regexp {[A-Z]} $ref] == 1} {
            set eref $url
            append eref [string tolower $ref]
            lappend urls $eref
        }
        set p1 [string first "<A HREF" $dir1]
    }
    return $urls
}

```

ibmime commands

Note: `ibmime` is included for backwards compatibility only.

`ibmime` is a Tcl extension package provided with Intelligent Broker for system implementers to:

- Programmatically, in Tcl UPoCs, to create or process messages that are exchanged between the engine and the Intelligent Broker server.
- Invoke remote web services (outgoing).

- Process incoming web service requests.

Client thread

An Intelligent Broker SOAP client thread invokes a web service hosted on a remote server. It does this by creating an `ibmime` message, populating its contents, and sending this message to the local Intelligent Broker server. It then listens for the reply from the Intelligent Broker server and processes any response.

Server thread

An Intelligent Broker SOAP server thread receives a web service request from a remote client. When the local Intelligent Broker server receives the request, it instantiates an `ibmime` message with appropriate content from the request. Then, it sends the message to the listening Intelligent Broker server thread. At this point, it processes the message and composes a reply to send back to the Intelligent Broker server.

ibmime Tcl API

Note: `ibmime` is included for backwards compatibility only. Help for the `ibmime` Tcl package is available in the online Tclhelp under `integrator/tcl/lib/cloverleaf/help/hci/ibmime`.

Before the `ibmime` extension can be used by a Tcl procedure, you must put `package require ibmime` in the start mode of the procedure.

This is a list of the Tcl methods used to create and access various parts of the `ibmime` message:

- `ibmimecreate`

This creates an empty IB Mime object. When called without any parameter, `ibmimecreate` creates an empty `ibmime` object whose content can then be populated later with the various API methods.

Example:

```
set ih1 [ibmimecreate]
```

To instantiate the input IB Mime message:

```
set ih1 [ibmimecreate $cont]
```

Note: The `ibmimecreate` method must be used, with the message received from Intelligent Broker as its parameter, to instantiate the `ibmime` object. This object is populated with the content of the message. This can then be passed to other `ibmime` API methods.

- `ibmimeheaderadd`

This adds protocol (HTTP, SMTP, POP3) headers.

Example:

```
ibmimeheaderadd $ih1 header_name_1 header_value_1  
ibmimeheaderadd $ih1 header_name_2 header_value_2
```

- `ibpartcreate`

This creates the first part of the IB Mime.

For example, this adds headers to the current part:

```
set phl [ibpartcreate
```

Example:

```
ibpartheaderadd $ih1 $phl "Content-Type" "text/xml"
```

- `ibpartcontentset`

This sets the content of the part.

Example:

```
# Set the content of the 1st part
ibpartcontentset $ih1 $phl soap_envelope
# create the 2nd part of this IB Mime
set phl [ibpartcreate $ih1]
# add headers of the 2nd part
ibpartheaderadd $ih1 $phl "Content-Type" "text/plain"
# add content of the 2nd part.
ibpartcontentset $ih1 $phl $cont
```

- `ibmimeencode`

This generates an IB MIME string from an `ibmime` object. `ibmimeencode` must be used to convert an `ibmime` object, created with the `ibmimecreate` method, to an actual `ibmime` message. This message is then sent to the Intelligent Broker server.

Example:

```
msgset $mh1 [ibmimeencode $ih1]
```

- `ibmimeheaderget`

This gets the protocol headers.

Example:

```
set header_value_1 [ibmimeheaderget $ih1 header_name_1]
set header_value_2 [ibmimeheaderget $ih1 header_name_2]
```

- `ibmimecountget`

This gets the number of parts in the MIME message.

Example:

```
set count [ibmimecountget $ih1]
```

- `ibpartheaderget`

This gets the header of a part.

For example, to get the header named `file_name` in the second part (Part ID = 1) of the MIME message:

```
set header_value [ibpartheaderget $ih1 1 "file_name"]
```

- `ibpartcontentget`

This gets the content of a part.

For example, the second part (Part ID = 1) content is the message:

```
set content [ibpartcontentget $ih1 1]
set content [ibpartcontentget $ih1 1]
```

- `ibmimeheadercountget`

This returns the number of headers in the main mime package.

Example:

```
ibmimeheadercountget $ih
```

- `ibmimeheadernames`

This returns a list of header names in the main mime package.

Example:

```
ibmimeheadernames $ih1
```

- `ibmimeheaderexists`

This returns a 0 or 1 if the header name exists in the main mime package.

Example:

```
ibmimeheaderexists $ih1 $name
```

- `ibmimeheadersget`

This gets all headers out, for example as a list of lists with two elements. This is similar to a keyed list, but avoids the risk that there is a dot in a key name.

Example:

```
ibmimeheadersget $ih1
```

- `ibmimesize`

This returns the size in bytes of the main mime package. If inbound, then this is equal to `[msglength $mh]`.

Example:

```
ibmimesize $ih1
```

- `ibpartheadercountget`

This returns the number of headers in the mime part.

Example:

```
ibpartheadercountget $ih1 $p
```

- `ibpartheadernames`

This returns a list of header names in the mime part.

Example:

- `ibpartheaderexists`

This returns 0 or 1 if the header name exists in the mime part

Example:

```
ibpartheaderexists $ih1 $p $name
```

- `ibpartsize`

This returns the size in bytes of the mime part.

Example:

```
ibpartsize $ih1 $p
```

- `ibpartheadersget`

This gets all headers out, for example as a list of lists with two elements. This is similar to a keyed list, but avoids the risk that there is a dot in a key name.

Example:

```
ibpartheadersget $ih1 $p
```

ibmime usage examples

These Tcl procs are part of the `ib.tcl` file that is part of the system distribution, and are used in various sample sites.

These are templates for taking in, sending out requests as client, and sending out responses as server, messages that are properly formed SOAP envelopes.

proc IbOut

This is an example of sending out SOAP messages that have an attachment containing the real message. The SOAP body contains only symbolic text to convey that the attachment has the real message.

In `IbOut`, messages coming into this procedure are not well-formed SOAP envelopes ready to access the first part of the `ibmime`. These are text or binary arbitrary messages. They access the attachment part, second part or beyond, of the `ibmime` object going out to IB. The SOAP envelope is manually added using hard-coded strings.

proc IbIn

This is an example of sending taking in SOAP messages that have an attachment containing the real message. The SOAP body contains only symbolic text to convey that the attachment has the real message.

In `IbIn`, messages coming into this procedure are considered to be an `ibmime` from IB. In the first part of the `ibmime`, it contains a SOAP envelope with only symbolic meaning. The attachment, the second part or beyond, carries the real message. The attached message is extracted from the `ibmime` and is forwarded on to the next thread for processing. This also assumes that a reply is expected by the sender, so a reply is generated and sent back.

proc IbOutXML

This is an example of sending out SOAP messages that carry the real message in the SOAP body.

In `IbOutXML`, messages coming into this procedure are in XML format that is ready to access the SOAP body of the envelope. For example, HL7 v3. The SOAP envelope is manually added using hard-coded strings. They then become the first part of the `ibmime` object going out to IB.

proc IbInXML

This is an example of taking in SOAP messages that carry the real message in the SOAP body.

In `IbInXML`, messages coming into this procedure are considered to be an `ibmime` from IB. These contain a SOAP envelope in the first part of the `ibmime`. This carries the real message in its SOAP body. For example HL7 v3. This procedure uses Tcl regular expressions to extract the XML message inside the SOAP body and forward it on for further processing.

proc IbEnvelopeIn

The `IbEnvelopeIn` proc, along with `IbClientEnvelopeOut` is a template for taking in messages that are properly formed SOAP envelopes. This is used within the sample sites that take advantage of the XML capabilities (XML formats) and translations, with the help of the XSD/WSDL tools. `IbServerEnvelopeOut`, as a group are considered a better approach to the SOAP/XML messaging solution than the `IbOutXML/IbInXML` pair because of better overall XML handling.

The hard-coded string and regular expression techniques that are used in that pair can be used with XML. When you do this, difficulties could happen when namespaces and sophisticated features are involved.

```
#####
# Name:                IbEnvelopeIn
# Purpose:              The message coming into this proc should be an ibmime
#                       structured message from IB. This proc will unwrap the
#                       soap envelope inside and replace the message's contents
#                       with this soap envelope. This can be used regardless
#                       of whether IB is acting as the client or server.
# UPoC type:           tps
# Args:                tps keyedlist containing these keys:
#                       MODE      run mode ("start", "run" or "time")
```

```

#                               MSGID    message handle
#                               ARGS      user-supplied arguments:
#                                       <describe user-supplied args here>
#
# Returns:                      tps disposition list:
#                               <describe dispositions used here>
#

proc IbEnvelopeIn { args } {
    keylget args MODE mode           ;# Fetch mode

    set dispList {}                  ;# Nothing to return

    switch -exact -- $mode {
        start {
            # Perform special init functions
            # N.B.: there may or may not be a MSGID key in args
        }

        run {
            # 'run' mode always has a MSGID; fetch and process it
            keylget args MSGID mh
            #First, parse the data to get the message type
            set cont [msgget $mh]
            # parse the input ib Mime message
            set ih1 [ibmimecreate $cont]
            # the 1st part (Part ID = 0) content is XML message
            set data [ibpartcontentget $ih1 0]
            # set the new message content as the inbound message
            msgset $mh $data
            lappend dispList "CONTINUE $mh"
        }

        time {
            # Timer-based processing
            # N.B.: there may or may not be a MSGID key in args
        }

        shutdown {
            # Doing some clean-up work
        }
    }

    return $dispList
}

```

proc IbClientEnvelopeOut

IbClientEnvelopeOut is a template for sending out requests as client messages that are properly formed SOAP envelopes. This is used within the sample sites that take advantage of the XML capabilities (XML formats) and translations, with the help of the XSD/WSDL tools.

This proc, along with IbEnvelopeIn and IbServerEnvelopeOut, as a group are considered a better approach to the SOAP/XML messaging solution than the IbOutXML/IbInXML pair. This is because of better overall XML handling.

The hard-coded string and regular expression techniques that are used in that pair can be used with XML. If this is used, then difficulties could happen when namespaces and sophisticated features are involved.

```

#####
#Name:                      IbClientEnvelopeOut
# Purpose:                  The message coming into this proc should contain an
#                           XML soap envelope string. The proc wraps this in an
#                           ibmime structure to send to IB, which IB will use to

```

```

#           invoke a web service. This proc should only be used
#           when IB is the client to a web service. If IB is the
#           web service server, then use the IBServerEnvelopeOut
#           proc.
#UPoC type:      tps
#Args:           tps keyedlist containing these keys:
#               MODE      run mode ("start", "run" or "time")
#               MSGID     message handle
#               ARGS      user-supplied arguments:
#                       <describe user-supplied args here>
#
#Returns:        tps disposition list:
#               <describe dispositions used here>
#
proc IbClientEnvelopeOut { args } {
    keylget args MODE mode                ;# Fetch mode

    set dispList {}                       ;# Nothing to return

    switch -exact -- $mode {
        start {
            # Perform special init functions
            # N.B.: there may or may not be a MSGID key in args
            package require ibmime
        }

        run {
            # 'run' mode always has a MSGID; fetch and process it
            global env
            keylget args MSGID mh
            set cont [msgget $mh]
            # create an empty ib Mime object
            set ih1 [ibmimecreate]
            # add ib Mime headers
            ibmimeheaderadd $ih1 "IBSite" $env(HCISITE)
            ibmimeheaderadd $ih1 "IBThread" [msgmetaget $mh DESTCONN]
            # create the 1st part of this ib Mime
            set ph1 [ibpartcreate $ih1]
            # add headers of the 1st part
            ibpartheaderadd $ih1 $ph1 "Content-Type" "text/xml"
            # add original message content as the 1st part.
            ibpartcontentset $ih1 $ph1 $cont
            msgset $mh [ibmimeencode $ih1]
            lappend dispList "CONTINUE $mh"
        }

        time {      # Timer-based processing      }

        shutdown {
            # Doing some clean-up work
        }
    }

    return $dispList
}

```

proc IbServerEnvelopeOut

IbServerEnvelopeOut is a template for sending out responses as server messages that are properly formed SOAP envelopes. They are used within the sample sites that take advantage of the XML capabilities (XML formats) and translations, with the help of the XSD/WSDL tools.

This proc, along with IbEnvelopIn and IbClientEnvelopeOut, as a group are considered a better approach to the SOAP/XML messaging solution than the IbOutXML/IbInXML pair. This is because of better overall XML handling.

The hard-coded string and regular expression techniques that are used in that pair can be used with XML. When you do this, difficulties could happen when namespaces and sophisticated features are involved.

```
#####
# Name:                ibServerEnvelopeOut
# Purpose:              The message coming into this proc should contain an
#                       XML soap envelope string. The proc wraps this in an
#                       ibmime structure to send to IB, which IB will use to
#                       send a web service reply. This proc should only be used
#                       when IB is the server of a web service. If IB is the
#                       web service client, then use the IBClientEnvelopeOut
#                       proc.
# UPoC type:           tps
# Args:                tps keyedlist containing these keys:
#                       MODE      run mode ("start", "run" or "time")
#                       MSGID     message handle
#                       ARGS      user-supplied arguments:
#                               <describe user-supplied args here>
#
# Returns:             tps disposition list:
#                       <describe dispositions used here>
#
proc IbServerEnvelopeOut { args } {
    keylget args MODE mode                ;# Fetch mode

    set dispList {}                      ;# Nothing to return

    switch -exact -- $mode {
        start {
            # Perform special init functions
            # N.B.: there may or may not be a MSGID key in args
            package require ibmime
        }

        run {
            # 'run' mode always has a MSGID; fetch and process it
            global env
            keylget args MSGID mh
            # get the soap envelope from the message
            set cont [msgget $mh]
            # create an empty ib Mime object
            set ih1 [ibmimecreate]
            # add ib Mime headers
            ibmimeheaderadd $ih1 "IBStatus" "APP_RESP"
            ibmimeheaderadd $ih1 "Content-Type" "multipart/mixed"
            # create the 1st part of this ib Mime
            set ph1 [ibpartcreate $ih1]
            # add headers of the 1st part
            ibpartheaderadd $ih1 $ph1 "Content-Type" "text/xml"
            # set the soap envelope as the part content
            ibpartcontentset $ih1 $ph1 $cont
            # update the message contents to the ibmime structure
            msgset $mh [ibmimeencode $ih1]
            lappend dispList "CONTINUE $mh"
        }

        time {
            # Timer-based processing
        }

        shutdown {
            # Doing some clean-up work
        }
    }

    return $dispList
}
```

Control headers between Cloverleaf and Intelligent Broker

Note: `ibmime` is included for backwards compatibility only.

The engine and Intelligent Broker exchange `ibmime` messages, as specified in the previous sections. There are a few MIME headers, not part headers, in each `ibmime` message. These have special meaning to the Intelligent Broker server. Therefore, they must be correctly set. Others are set by the Intelligent Broker server and convey special status information and therefore should be examined by the Tcl procedures.

The headers that are expected by Intelligent Broker on an outgoing `ibmime` request message by a client thread are:

- `IBSite`
- `IBThread`
- `Content-Type`

The headers that are expected by Intelligent Broker on an outgoing `ibmime` reply message by a server thread are:

- `IBStatus`
- `Content-Type`

The header set by Intelligent Broker on an incoming `ibmime` reply message to a client thread is `IBStatus`.

There are no headers set by Intelligent Broker on an incoming `ibmime` request message to a server thread.

Basically, `IBSite` and `IBThread` headers identify to Intelligent Broker which client site/thread sends this `ibmime` request. Then, Intelligent Broker can look up configured information about this site/thread and act correctly. On the other hand, `IBStatus` represents a status report back to the client thread about the request being sent to the remote service.

`IBStatus` also serves a dual role when a server thread is sending a reply to a request. It is used as a directive to the Intelligent Broker server. It determines the type of response that is sent back to the remote client.

IB client (outbound) thread: Normal message flow

For an IB client (outbound) thread, the normal message flow for the request/response messages between an Intelligent Broker client and a server is:

- 1 The Intelligent Broker client thread composes an `ibmime` message and sends it to Intelligent Broker which transmits it to the remote endpoint.

The `ibmime` message must have these headers:

- `IBSite` and `IBThread`, which identify the thread sending the message.
- `Content-Type`: This is "multi-part/mixed" because messages can have attachments. Even without attachments, the message is still a multi-part message with a single part, that is, the message body itself.

- 2 Intelligent Broker sends the message to the remote endpoint, as configured in the thread properties.
- 3 When the remote endpoint sends back a response, that reply is forwarded back to the IB client thread.

The Intelligent Broker client thread in the system must select the **Await Replies** check box under Inbound Replies of the thread properties **Inbound** tab. This must be selected to receive the reply from the remote

endpoint, forwarded by Intelligent Broker. If the box is not selected, then any reply from the remote endpoint is not forwarded by Intelligent Broker. The forwarded message is an `ibmime` message with a distinct `ibmime` header, `IBStatus` with the value `APP_RESP`, indicating this is a response.

The WSDL for the remote endpoint may specify that it is a request/response type of service, or a one-way type. The latter means that the web service does not return meaningful data as a reply. It is only an indication that it has received the request. The `ibmime` message sent back to the Intelligent Broker client, by Intelligent Broker, has a distinct header `IBStatus` with the value `APP_ACCEPT`. This indicates the request has been accepted by the remote end. Otherwise, no reply is received.

To receive this `APP_ACCEPT`, the thread that originates the request must select the **Await Replies** check box. This requires that Intelligent Broker always send back a reply, even when the remote web service is one-way, per its WSDL.

When that box is not selected, Intelligent Broker does not send any reply back to the thread originating the request. It does not matter that the requested web service happens to be of request/response type per its WSDL.

The `IBStatus` of `APP_RESP` and `APP_ACCEPT` represent normal responses from the remote endpoint.

Note: Normal business flow can include conditions where a reply indicates that the request cannot be fulfilled. For example, a withdrawal from a bank account cannot be granted because the balance is too low, or the account number is incorrect. These are all considered by Intelligent Broker as a normal reply and therefore `APP_RESP` is returned to the system.

IB client (outbound) thread: Error conditions

When an error happens, Intelligent Broker populates the `IBStatus` of the `ibmime` message with the values that are listed in this topic. It also includes a detailed error description in the `ibmime` message body.

The Intelligent Broker log files also have detailed information about the error:

- `APP_REJECT`

An exception could happen which is not an application-level error according to the business logic. For example, the remote service finds that the requested resource cannot be granted. A system or network level error could happen, so the remote endpoint is inaccessible by the local Intelligent Broker. For example, the remote endpoint URL that is given in the Intelligent Broker client thread properties is not a valid URL. The `IBStatus` that is returned by the local Intelligent Broker then has the value of `APP_REJECT` with a subcode a /.

For example, `APP_REJECT/HXML_47`.

The message body contains a detailed error description.

Common sub-codes `APP_REJECT` include:

- `IB_00`: Reason not known. The error description might contain more detail.
- `IB_01`: Indicates that the remote endpoint is not running or otherwise not accessible. This indicates that it is not permitted or the resource is not found on the endpoint.
- `IB_02`: The remote host indicated in the endpoint URL does not exist.
- `IB_03`: The message being sent is not well-formed XML.
- `IB_04`: The message being sent does not have a valid soap Envelope.

- **IB_05:** Indicates the remote endpoint has not sent back a response in a preset period of time. This could result if the IP does not respond at all or is merely taking more time to process than permitted.
- **IB_06:** The outbound message has an attachment with no content type header.
- **IB_07:** A non-integer was passed in the `IBSoapSendTimeout` header.
- **IB_08:** The destination is missing protocol, most likely an invalid site/thread or `IBEndpointURL` header is missing `http://` part.
- **COMMIT_ERR**
 Certain errors are not caused by system or network anomalies, in which case the `IBStatus` in the reply contains `COMMIT_ERR`. For example, There are software problems or otherwise unknown reasons.
 An outbound thread making a request/response web service invocation can get back `APP_RESP`, `APP_REJECT`, or `COMMIT_ERR`.
 A one-way web service invocation can get back `APP_ACCEPT`, `APP_REJECT`, or `COMMIT_ERR`.
 Among all of these replies, only the `APP_RESP` reply has a meaningful content associated with the rest of the `ibmime` structure.

IB server (inbound) thread: Normal message flow

When Intelligent Broker receives a request, on behalf of an Intelligent Broker server thread, from a remote client, Intelligent Broker instantiates an `ibmime` message. This message represents the received request, and is forwarded to the server thread.

The server thread must compose and return an `ibmime` message to Intelligent Broker. This reply `ibmime` must have a distinct header `IBStatus`. This serves as a directive to its Intelligent Broker as to how to reply to the remote client, which can take one of two values:

- **APP_RESP:** Indicates that the server thread, the web service, has processed the request and is returning a meaningful reply. The `IBStatus` must be set when the request results in an error condition in the web service and the reply by the web service includes a SOAP fault. It is considered `APP_RESP` by Intelligent Broker.
- **APP_ACCEPT:** If the web service, represented by the server thread, is a one-way service, then the `IBStatus` header returned to Intelligent Broker must have this value. This instructs Intelligent Broker to send back an acknowledgment that is compliant with W3C standards.

IB server (inbound) thread: Error conditions

When an error happens on the Intelligent Broker server side, it usually involves one of these issues:

- The incoming request is missing important information such as the site/thread of the destination endpoint.
- The incoming request is not a properly formed SOAP message, or even XML, or otherwise is not recognizable as a valid web service request.
- The local Intelligent Broker is not properly configured, therefore it cannot effectively deliver the request to the Intelligent Broker server thread for processing.
- The Intelligent Broker server thread is not running, so the delivery by Intelligent Broker to that thread failed.

ibmime headers overriding default behavior

Note: `ibmime` is included for backwards compatibility only.

The Intelligent Broker client thread in the system generally sends a message to an endpoint that is statically configured in the thread properties. Intelligent Broker sends the messages out with preset guidelines.

In some scenarios, it is desirable for the client thread to override that default behavior. This is because it dynamically deems appropriate when sending a particular `ibmime` message to Intelligent Broker for forwarding to the remote endpoint. This is performed using these specific `ibmime` control headers.

IBEndpointURL and IBSOAPAction

These override the SOAP endpoint and the SOAPAction associated with the operation to invoke, for the client thread as configured in the thread properties. This is only for this instance of the message.

When the endpoint URL indicates the endpoint address, the SOAPAction indicates the operation on that endpoint to invoke. If the endpoint URL is changed, then the SOAPAction header must be changed as well.

Tcl example for setting the header using the `ibmime` API:

```
ibmimeheaderadd $ih "IBEndpointURL" http://SomeOtherHost:8080/abc
ibmimeheaderadd $ih "IBSOAPAction" http://somehost/someaction
```

IBSoapSendTimeout

Intelligent Broker sends the SOAP message to the remote endpoint. Then it waits for a preset number of seconds for a reply before returning an error `ibmime` message. The default is 60 seconds. This is sent to the client thread (`IBStatus = APP_REJECT`).

The client can override this time-out for this particular message to be longer or shorter.

For example, using Tcl to set the `ibmime` header for a time-out of 2 minutes (120 seconds or 120000 milliseconds):

```
ibmimeheaderadd $ih "IBSoapSendTimeout" "120000"
```

The value is in milliseconds. This must be between the range of 0 and 3600000 inclusive, where 3600000 is 10 hours, and 0 is not waiting at all.

Sample Tcl procedures

Note: `ibmime` is included for backwards compatibility only.

An `ib.tcl` file under the `tclprocs` directory of the system installation contains the procedures used in the sample sites. These procedures provide an example of how to create and process the `ibmime` messages. These messages are exchanged at run time between the engine and the Intelligent Broker server.

These procedures are:

- **IbIn:** When receiving an inbound `ibmime` message, `IbIn` extracts the second part of the message (SOAP or email attachment). Then it continues with the extracted part as the new message.

- **IbOut:** This creates an `ibmime` message, marks the current message as the SOAP attachment of this new `ibmime` message, and continues with the new `ibmime` message.
- **IbInXML:** When receiving an inbound `ibmime` message, `IbInXML` extracts the first part of the message and continues with the extracted part as the new message.
- **IbOutXML:** This creates an `ibmime` message, makes the current message as the SOAP first part of the `ibmime` message, and continues with the new `ibmime` message.
- **IbEnvelopIn, IbClientEnvelopeOut, IbServerEnvelopeOut:** These are templates for taking in and sending out requests as client. They are also for sending out responses as server messages that are properly formed SOAP envelopes.

Tcl templates

Two Tcl procedure types are templates that help with the `ibmime` objects:

- The `ibin` Tcl template helps developers to parse `ibmime` and retrieve the message data.
- The `ibout` Tcl template helps developers to wrap the message data into `ibmime` format.

These Tcl templates are available on the Script Editor, located on the **Proc Type** list of the **New Proc** dialog box.

Tcl extensions

Tcl extensions enable message manipulation within the engine.

For additional information, numerous web sites are devoted to Tcl, including the Tcl Developer Xchange. See www.tcl.tk.

Conventions

Command lines often contain special characters:

- `?` (optional)
- `|` (or)
- `...` (and so on)
- `-` (options)
- *variable* (required variable)

Engine information

The `engprotostate` command is only valid in a Tcl interpreter within a protocol thread. It returns the protocol state of the current thread as a readable string (initializing, opening, up, down, closing, error, or ineof).

Counter commands and references

Counter commands are Tcl procedures for counter objects, which generate sequential numbers within a range.

All counter procedures accept a mode argument. The system accepts only `file` as a value.

CtrlInitCounter

This command initializes a counter. Counters store their configurations and states in files.

When creating a counter, specify these characteristics:

- The name for the counter file. The required `.CTR` extension is automatically appended to the end of the file name.
- The initial value. This defaults to 1.
- The maximum value. This defaults to 999999999.
- The reset value. This defaults to 1.

`CtrlInitCounter` returns the configured start, max, and reset values.

For example, this command initializes a counter that handles the numbers from 1 to 3:

```
Input: CtrlInitCounter CTR <file name> [1] [3]
Output: 1 3 1
```

A file named `file name.CTR` is created by the system in the local directory. To operate on this counter with other CTR procedures, specify a file path that identifies this file without the `.CTR` extension.

To initialize a counter file with specified parameters, use:

```
CtrlInitCounter tag ?mode? ?start? ?max? ?reset?
```

- `tag` is the counter file path.
- `mode` is the file.
- `start` is the initial counter value. This defaults to 1.
- `max` is the maximum counter value. This defaults to 999999999.
- `reset` is the reset counter value. This defaults to 1.

CtrlNextValue

When using a counter, the usual case is to query and update the counter. This command retrieves the counter's current value, increments it, updates the file, and returns the original value.

For example:

```
Input: CtrlNextValue CTR
Output: 1
```

```
Input: CtrNextValue CTR
Output: 2
```

When updating a counter causes it to exceed its configured maximum value, it rolls over to its reset value.

For example:

```
Input: CtrNextValue CTR
Output: 3
Input: CtrNextValue CTR
Output: 1
```

To retrieve the counter's current value, increment the counter's value, and change to its reset value. If required, then use:

```
CtrNextValue tag ?mode?
```

- tag is the counter file path.
- mode is the file.

CtrCurrentValue

This command is used to query a counter's value without updating it.

For example:

```
Input: CtrCurrentValue CTR
Output: 2
Input: CtrCurrentValue CTR
Output: 2
```

To retrieve the counter's value without updating it, use:

```
CtrCurrentValue tag ?mode?
```

- tag is the counter file path.
- mode is the file.

CtrResetValue

This command is used to set a counter to its reset value and return an empty string.

For example:

```
Input: CtrResetValue CTR
```

If the named counter file does not exist, CtrResetValue creates a new counter with default values.

For example:

```
Input: CtrResetValue NEW_CTR
Output: 1 999999999 1
```

A counter is destroyed by removing the counter file.

For example, in UNIX:

```
% rm NEW_CTR.ctr
```

In Windows:

```
del NEW_CTR
```

To change the counter's value to its configured reset value, use:

```
CtrResetValue tag ?mode?
```

- `tag` is the counter file path.
- `mode` is the file.

Datum extensions and references

The Tcl datum extensions perform operations on a message fragment, known as a datum. Datum extension objects associate a system-supported data type with arbitrary data. They are most often used in conjunction with GRM and XPM handles. See [GRM extensions](#) and [XPM extensions](#).

Datum (DAT) objects are the smallest unit of data available in a given record format. For example, in FRL, a datum represents a subfield; in HL7, it represents a subcomponent.

Datum objects have two attributes:

- `VALUE`
- `TYPE`: The type specifies the nature of the value. It is particularly useful for parsing and converting date and time values. A datum's type must be a legal hci datatype. : A datum's value is the characters or number the datum represents.

Although the type value must be a legal datatype, no type checking is performed against the value. It is therefore up to the user to ensure a datatype value matches its type. Unexpected recalls can happen with mismatched types and values.

This section gives a brief overview of the commands that permit manipulation of datum (DAT) objects. Datum objects are most useful when used in conjunction with other hci extensions. Datum objects are included in examples. See [GRM extensions](#) and [XPM extensions](#).

Datum objects represent empty values as empty value strings. They represent active null values with a type value of null. Record formats use and identify active null values differently.

Datum objects

Legal datum objects are:

- `ai`: ASCII integer; optional sign and digits (0-9)
- `ch`: string/character data. This is similar to `st`.
- `dt`: `?cc?yymmdd`
- `ed`: date; `ddmm?cc?yy`
- `fd`: `mm/dd/?cc?yy`
- `fe`: `dd/mm/?cc?yy`
- `jd`: `?cc?yy?ddd` (Julian)
- `nm`: numeric; optional sign, digits, optional decimal point
- `null`: present, but empty value
- `st`: string/character data. This is similar to `ch`
- `tm`: `hhmm[ss][+/-zzzz]` (time plus optional timezone)
- `ts`: `ccyymmddhhmm[ss][+/-zzzz]`
- `ut`: Day Mon dd hh:mm:ss ccyy
- `yd`: `mmdd?cc?yy`

Creating datum objects

Datum objects are created when data is retrieved through GRM or XPM handles. See [GRM extensions](#)[XPM extensions](#).

They are also created with the `datcreate` command. `datcreate` creates one datum at a time and returns the new datum's handle.

If `datcreate` is not give any arguments, then a datum representing an empty field is created. and

For example:

```
andInput: datcreate
Output: datum0
```

To create datum with a particular value:

```
Input: datcreate "some data"
Output: datum1
```

To specify the datum's datatype:

```
Input: datcreate 17760705 dt
Output: datum2
```

If not specified, it defaults to `ch`.

Viewing datum handles

The `datlist` command shows the valid datum handles currently available.

For example, first create new datum:

```
Input: datcreate; datcreate; datcreate
Output: datum0 datum1 datum2
```

To view the newly-created datum:

```
Input: datlist
Output: datum0 datum1 datum2
```

Destroying datum

Remove datum from the interpreter with the `datdestroy` command.

Whether `datcreate` or `grmcreate` is used to create a datum, that datum remains in the interpreter until it is destroyed. As long as it is in the interpreter, it uses a certain amount of memory.

The `datdestroy` command frees the resources of one or more datum objects. This command has two forms: one takes one or more datum handles as individual arguments; the other takes a single list of handles.

To destroy all of the datum objects in an interpreter:

```
datdestroy -list [datlist]
```

For example, these commands are equivalent:

```
datdestroy datum0 datum1 datum2
datdestroy -list "datum0 datum1 datum2"
```

The `hcidatlistreset` procedure, supplied in the system library, permits all but a particular set of datum handles to be destroyed. This command's only argument is the list of datum handles to preserve.

For example:

```
set datSaveList [datlist];# record existing handles
# Any amount of code which creates datum objects
# to be destroyed at the end of the section.
set newDatId [datcreate]
hcidatlistreset $datSaveList
```

Datum object characteristics

The `datget` and `datset` commands provide a keyed-list interface to datum object characteristics. If `datget` is called with only a datum handle, then it returns the characteristic keys that are available.

For example:

```
Input: datcreate 17760704 dt
Output: datum2
Input: datget datum2
Output: TYPE VALUE
```

Use these keys to query the characteristics.

For example:

```
Input: datget datum2 TYPE
Output: dt
Input: datget datum2 VALUE
Output: 17760704
```

Modify the datum through `datset`.

datcreate

```
datcreate ?VALUE? ?TYPE?
```

This command creates a new datum of the specified `VALUE` and `TYPE`.

- If specified, then `VALUE` is the string value of the new datum.
- `TYPE` is an hci supported data type. For example, `ai` or `ch`.
- New datum `VALUE` defaults to empty and `TYPE` `ch`.

This command returns the new datum handle for use by other commands.

datdestroy

```
datdestroy -list list
datdestroy datId1 datId2 ...
```

This command destroys one or more datum objects.

- If the `-list` option is used, then it contains the datum handles to destroy.
- Without the `-list` option, `datId1`, `datId2`, and so on, are the individual datum handles to destroy.

This command returns an empty string.

datget

```
datget datId ?key?
```

If a key is not specified, then the object's attribute keys are returned.

If a `key` is specified, then that attribute's value is returned.

datlist

```
datlist
```

This command returns a list of valid datum handles currently available in the Tcl interpreter.

hcidatlistreset

This procedure, supplied in the system library, permits all but a particular set of datum handles to be destroyed. This command's only argument is the list of datum handles to preserve.

This command returns an empty string.

datset

```
datset datId key VALUE ?key2 VALUE2...?
```

This command sets the datum attributes to the specified values.

- For `key`, use valid datum attribute keys.
- When setting the `TYPE`, use valid hci data types.
- No checking or verification is performed on the value.

GRM extensions and references

The Generic Record Manager (GRM) is the interface to all record format facilities of the system. The Tcl extensions in the GRM category permit access to record/message formats supported by the engine. They also permit access to symbolic message data, for example, by field name. Message objects hold raw data. To access a particular portion of the data, you must know its offset and length.

Specific record format rules assign meaning to the data:

- The FRL module distinguishes between field data and pad/fill characters.
- HRL uses field delimiters.
- XML uses tags.
- VRL uses tags or field delimiters.
- The HL7, UN/EDIFACT, NCPDP, and X12 modules recognize delimiter characters.
- All modules assign names to various portions of the data.

GRM objects are used for record parsing, data lookup, field modification, and message encoding. Using the Tcl extensions, or commands, described in this topic also gives you direct access to those same functions.

A working knowledge of these is required:

- FRL, VRL, HRL, XML, HL7, UN/EDIFACT, NCPDP, and X12 record formats, processing rules, field naming, and addressing.
- Message and datum objects and the commands to manipulate them.

Before using any of the GRM commands in a stand-alone Tcl interpreter, that is, an interpreter external to the engine, set your environment. This is accomplished using `setroot` and `setsite`. Your code must also invoke `setHciDirs` before accessing any of the GRM commands.

Creating GRM objects

`grmcreate` initializes a new GRM object and returns its handle. When a GRM object is created, it is assigned a type. This general type identifies the realm of record types the object manipulates. Legal types are FRL (frl), VRL (vrl), HRL (hrl), XML (xml), HL7 (hl7), UN/EDIFACT (edifact), NCPDP (ncpdp), and X12 (x12).

In FRL, a GRM object handles only one FRL definition at a time. When an FRL GRM is created, supply the name of the FRL definition to use.

For example:

```
Input: grmcreate frl my.frl
Output: grm0
```

In HL7, access requires three parameters: version, variant, and message type.

For example:

```
Input: grmcreate hl7 2.1 my-variant ADT_A01
Output: grm1
```

Use an empty string in place of the variant name to access the base HL7 v2.1 definitions.

For example:

```
Input: grmcreate hl7 2.1 {} ADT_A01
Output: grm2
```

For message parsing, `grmcreate` can parse a message object's contents immediately. Use the `-msg` option to specify the message to use.

For example:

```
Input: grmcreate -msg message0 frl my.frl
Output: grm3
```

The GRM object copies the message's contents and parses them according to the record definition. Modifying or destroying the message does not disturb the GRM object.

If the record type specifies any parse-time data validation (for example, FRL), then collect validation warnings with the `-warn` option. The interpreter places any warnings in the specified variable name.

For example:

```
Input: grmcreate -msg message0 -warn w frl my.frl
Output: grm4

Input: echo $w
Output: {TSFIELD: not valid ts data} {AIFIELD: not valid ai data}
```

Available handles

The `grmlist` command shows the GRM handles in the interpreter.

For example:

```
Input: grmlist
Output: grm0 grm1 grm2 grm3 grm4
```

Destroying GRM handles

You can destroy a GRM object at any time. Destroying a GRM object removes its handle from the interpreter and frees up the memory associated with the object. The `grmdestroy` command destroys any number of GRM handles, returning an empty string.

```
Input: grmlist
Output: grm0 grm1 grm2 grm3 grm4

Input: grmdestroy grm0 grm1 grm2
Output:

Input: grmlist
Output: grm3 grm4
```

Destroying a GRM handle does not affect any messages it has parsed or encoded, any retrieved datum objects, or any objects in the interpreter.

Querying field data

GRM objects give symbolic access to message data. `grmfetch` takes an address string and returns DAT objects containing the requested field data. Use the `-warn` option with `grmfetch`. You can also use `-warn` with `grmcreate`, `grmreset`, and `xpmfetch`.

Examples

To retrieve FRL data, assume `$grmIdFrl` is the GRM handle for a parsed FRL message record containing a NAME field with two subfields. For example:

```
Input: grmfetch $grmIdFrl NAME
Output: datum0 datum1
```

To retrieve subfields in definition order, the DAT handles returned correspond to the retrieved subfields. If the NAME subfields are first and last name, respectively, then `datum0` contains the first-name data and `datum1` contains the last name.

For example:

```
Input: datget datum0 VALUE
Output: Firstname
Input: datget datum1 VALUE
Output: Lastname
```

To retrieve specific subfields, request specific subfields by enclosing them in square-brackets. Because these characters usually have a special meaning in Tcl, surround the entire address in curly braces.

For example:

```
Input: grmfetch $grmIdFr1 {NAME.[1]}
Output: datum2

Input: datget datum2 VALUE
Output: Lastname
```

Retrieve specific subfields in any order by separating the subfield numbers with commas.

For example:

```
Input: grmfetch $grmIdFr1 {NAME.[1,0]}
Output: datum3 datum4

Input: datget datum3 VALUE
Output: Lastname

Input: datget datum4 VALUE
Output: Firstname
```

`grmfetch` returns an error when the field address does not match the record definition.

For example:

```
Input: grmfetch $grmIdFr1 NO_SUCH_FIELD
Output: Error: unable to resolve address "NO_SUCH_FIELD..."

Input: grmfetch $grmIdFr1 {NAME.[2]}
Output: Error: unable to resolve address "NAME.[2]"

Input: grmfetch $grmIdFr1 {NAME.[1,2]}
Output: Error: unable to resolve address "NAME.[2]"
```

When parsing and retrieving from a message object, `grmfetch` generates an error when a field's starting offset is beyond the end of the message object.

To retrieve HL7 data, in this example `$grmIdHL7` is the GRM handle for a parsed HL7 v2.1 message record containing a PID segment.

For example:

```
Input: grmfetch $grmIdHL7 0(0).PID(0).00041
Output: datum5 datum6 datum7

Input: datget datum5 VALUE
```

```
Output: Lastname
```

```
Input: datget datum6 VALUE
```

```
Output: Firstname
```

```
Input: datget datum7 VALUE
```

```
Output: MI
```

Order of retrieval

Retrieve component values in any order.

For example:

```
Input: grmfetch $grmIdHL7\ {0(0).PID(0).00041(0).[1,2,0]}
```

```
Output: datum8 datum9 datum10
```

```
Input: datget datum8 VALUE
```

```
Output: Firstname
```

```
Input: datget datum9 VALUE
```

```
Output: MI
```

```
Input: datget datum10 VALUE
```

```
Output: Lastname
```

Active null value

Specify HL7 subcomponents using another square-bracketed comma-separated list. If the component or subcomponent list has multiple comma-separated elements, then the other list cannot contain them.

HL7 uses double-quote characters to represent the active null value. This is a value that is present in the message but explicitly empty. When an active null (sub)component is retrieved, the resulting DAT object has TYPE null.

For example, assume the patient-name field has " " in its middle-initial component:

```
Input: grmfetch $grmIdHL7\ {0(0).PID(0).00041(0).[2]}
```

```
Output: datum11
```

```
Input: datget datum11 VALUE
```

```
Output:
```

```
Input: datget datum11 TYPE
```

```
Output: null
```

`grmfetch` errors when retrieving an illegal field path. For example, undefined segment names or field IDs. If the address is legal, but not represented in the GRM object or its parsed message, then `grmfetch` returns a DAT object with an empty value.

Modifying field data

Modify field data through the `grmstore` command. Similar to the `grmfetch` command, it can accept a single address string specifying multiple destination addresses.

After expanding the address list, it assigns field data values to those addresses.

`grmstore` returns an empty string.

Command forms

There are two `grmstore` command forms:

```
grmstore -list grmId field type arg
grmstore grmId field type arg ...
```

In the first form, `arg` is the list of values to be stored. The `-list` flag identifies `arg` as a list.

In the second form, the values are listed individually.

Input values

There are three types of input values:

- Raw character strings
- DAT object handles
- Special-values

The input values list can contain any mix of these elements. It can be a single list, first form, or discrete command-line elements, second form.

The type parameter declares the input value list's default element type:

- `c`: Character string
- `d`: Datum handle
- `v`: Special value string

Use other types by preceding them with an identifier.

`null` is the only special-value string supported. It represents the active null value.

`-type` is used with `grmstore` using character-string (for example, not special-value or datum) input list elements.

By default, character-string data are treated as `ch` data.

If `-type` is used, then specify any GRM data type. For example, any value usable with `datcreate`.

For example, these are equivalent:

```
Input: grmstore $grmIdHl7 0(0).PID(0).00041 c Last\ First MI
Output:

Input: grmstore -list $grmIdHl7 0(0).PID(0).00041\ c "Last First MI"
Output:
```

In both cases, `grmstore` assigns the strings Last, First, and MI, in order, to the target field's first three components.

These commands set a field value that is encoded as `Last^First^"`.

```
Input: datcreate First
Output: datum0
```

```
grmstore $grmIdHL7 0(0).PID(0).00041 c Last -d datum0 -v null
```

Input values are matched one-to-one to target addresses. If there are more addresses than input values, then the unmatched addresses are not modified; unmatched input values are ignored without error message.

Copying data

If there are two GRM handles similar in type, then use the `grmbulkcopy` command to copy data from one handle to the other.

`grmbulkcopy` returns an empty string.

For additional information, see [grmbulkcopy](#).

If the handles are HL7 objects, then all of the source object's data paths are duplicated in the target handle.

When copying data from one FRL object to another, only field names that are shared by both record definitions are copied.

Similar to `grmcreate`, `grmbulkcopy` can produce data validation warning messages when parsing the underlying message.

`grmbulkcopy` can also handle the `-warn` option.

For example:

```
grmbulkcopy -warn w $grmIdHL7Dest $grmIdHL7Src
grmbulkcopy -warn w $grmIdFrlDest $grmIdFrlSrc
```

Encoding message data

Use the `grmencode` command to encode a GRM object's field data into a properly formatted message, and return the newly encoded message handle.

The encoding process could produce some data validation warning messages. Use the `-warn` option to specify a variable to collect these messages.

For example:

```
Input: grmencode -warn w $grmIdFrl
Output: message0

Input: grmencode -warn w $grmIdHL7
Output: message1
```

Note: The messages that are produced by `grmencode` have minimal metadata values.

Resetting

Use the `grmreset` command to clear a GRM object's data state. It removes all of the field data values, but leaves the definition intact and ready for use. It returns an empty string.

For example:

```
grmreset $grmId
```

Use this command to parse a new message into the GRM object. When parsing a new message, `grmreset` supports a `-warn` option such as `grmcreate`.

For example:

```
grmreset -warn w $grmId $msgId
```

grmbulkcopy

This command copies record data from the source to the destination.

For example:

```
grmbulkcopy ?-warn var? destGrmId srcGrmId
```

This copies all of the record data from `srcGrmId` to `destGrmId`. The GRM objects must be the same type. For example, FRL, VRL, HRL, XML, HL7, X12, NCPDP, or UN/EDIFACT.

Data are copied from the source to the destination according to these types:

- UN/EDIFACT
Input data paths are copied directly to the output side unchanged. After the data are copied, they are available for retrieving from the output side.
When the data are encoded into a message, only the relevant paths are used.
- NCPDP
Input data paths are copied directly to the output side unchanged. After the data are copied, they are available for retrieving from the output side.
When the data are encoded into a message, only the relevant paths are used.
- FRL
Only field names common to both the input and output sides are copied. When copying each field, the subfields are matched one-to-one until one or both lists terminate.
- VRL
Only field names common to both the input and output sides are copied. When copying each field, the subfields are matched one-to-one until one or both lists terminate.
- HRL
Only field names common to both the input and output sides are copied. When copying each field, the subfields are matched one-to-one until one or both lists terminate.

- XML
Only field names common to both the input and output sides are copied. When copying each field, the subfields are matched one-to-one until one or both lists terminate.
- HL7
Input data paths are copied directly to the output side unchanged. After the data are copied, they are available for retrieving from the output side.
When the data are encoded into a message, only the relevant paths are used.
When copying between versions, for example, 2.1 and 2.2, a reasonable effort is made to map analogous standard fields.
User-defined field IDs are mapped directly. This assumes the same field ID is used in the same segment in both the input and output definitions.
- X12
Input data paths are copied directly to the output side unchanged. After the data are copied, they are available for retrieving from the output side.
When the data are encoded into a message, only the relevant paths are used.
If a warning variable name is specified, then any warning messages that are generated during the operation are collected in the named variable.
This command returns an empty string.

grmcreate

This command creates a new GRM handle for a record format.

```
grmcreate ?-msg msgId? ?-warn var? type args.
```

args parsing is based on type:

- DB

```
dbconnection tableschema ?tableschema ...?
```

dbconnection is the name of the database connection.

tableschema is the name of the table schema.

- FRL

```
frlname
```

frlname is the name of the FRL to use.

- VRL

```
vrlname
```

vrlname is the name of the VRL to use.

- HRL

```
hrlname
```

hrlname is the name of the HRL to use.

- HL7

```
vers variant msgType
```

vers is the version.

variant is the variant name.

msgType is the HL7 message type.

- HPRIM

```
vers variant msgType
```

vers is the version.

variant is the variant name.

msgType is the HPRM message type.

- LDL

```
vers variant msgType
```

vers is the version.

variant is the variant name.

msgType is the LDL message.

- NCPDP

```
vers variant msgType
```

vers is the version.

variant is the variant name.

msgType is the NCPDP message type.

- NCPDBFAB

```
vers variant msgType
```

vers is the version.

variant is the variant name.

msgType is the NCPDBFAB message type.

- NCPDPSCRIPT

```
vers variant msgType
```

`vers` is the version.

`variant` is the variant name.

`msgType` is the NCPDPSCRIPT message type.

- XML

```
package ocmname ?rootnode?
```

`package` is the XML package.

`ocmname` is the name of OCM to use.

`rootnode` is the root node of the OCM. This is optional.

- X12

```
vers variant msgType
```

`vers` is the version.

`variant` is the variant name.

`msgType` is the X12 message type.

- UN/EDIFACT

```
vers variant msgType
```

`vers` is the version.

`variant` is the variant name.

`msgType` is the UN/EDIFACT message type.

If a valid `msgId` is supplied, then that message is available to the GRM handle for parsing and data retrievals.

If a warning `var` variable name is specified, then any warning messages that are generated during the operation are collected in the named variable.

This command returns a new GRM handle for use in other commands. Use multiple GRM handles to access the same definition in separate messages without interfering with each other.

grmdestroy

This destroys the specified GRM handles.

```
grmdestroy grmId ?...?
```

This command returns an empty string.

grmencode

This encodes the data stored in the handle according to the handle's format. The data state of the GRM handle remains unchanged.

```
grmencode ?-sepchars keylist? ?-meta metadata?  
?-class class? ?-type type? ?-recover? ?-warn var? grmId
```

If a warning variable name is specified, then any warning messages that are generated during the operation are collected in the named variable.

By default, this command returns the newly created message handle of class `ENGINE`, type `DATA`, and with the `USERDBRECOVER` flag off.

grmlist

This returns a list of valid GRM handles currently available in the Tcl interpreter.

For example:

```
grmlist
```

grmreset

This clears the data state. If `msgId` is specified, then this command also parses that message.

For example:

```
grmreset ?-warn var? grmId ?msgId?
```

If a warning variable name is specified, then any warning messages that were generated during the operation are collected in the named variable.

This command returns an empty string.

grmstore

This stores one or more values in a GRM handle's data-state according to field. If more values are supplied than are implied by field, then the extra values are ignored.

```
grmstore ?-list? grmId field type args  
grmstore grmId field type args...
```

- If the `-list` option is used, then `args` is a list containing all of the values to store. Otherwise, the value list is all of the remaining arguments to the command.

- The type value, `c`, `d`, or `v`, identifies the default item type that the list uses, although the datum can show without identification.
- Constant strings use `-c`; datum objects use `-d`; special values use `-v`.
- Constant string values default to type `ch` when stored.
- Store both strings and datum objects using this command.
- The only special value currently supported is `null`, which represents a value that is present but empty. In HL7, for example, this is encoded as `""` (two double-quotes); it is not treated specially in FRLs.
- Specify an hci data type with a `-type` data type trailer. For example, `-c string -type datatype`.

Other examples are:

```
grmstore grm0 PatientName d datum0 datum1
grmstore -list grm0 {PatientName.[0,1]} d "datum0\ datum1"
grmstore grm1 0(0).PID(0).00041 d -c Smith datum2\ -v null
```

This command returns an empty string.

grmpathinfo

This returns the group info into which the segment is grouped. `segId` is the line number of the segment in the message definition, not including the prologue. The first line index is "0."

```
grmpathinfo grmId segId
```

For example, a user must find which group segment IN2, whose line number is 27, of ADT_A01 with version 2.5 is grouped into.

To do this, the user can run `grmpathinfo $mh 27` in a `hcitcl` shell:

```
set grm [grmcreate -msg $mh hl7 2.5 {} ADT_A01]
set pathinfo [grmpathinfo $grm 27]
puts $ pathinfo

Result: 5 0
```

This identifies the group path of IN2, that is, the parse result address of IN2 is similar to `5(0).0(0).IN1(0)`.

International character data support

Tcl, which is embedded within the system, supports international character data. It does this by manipulating string data internally in Unicode, specifically, UTF-8 or 8-bit Unicode Transformation Format. This permits support for numerous single-byte and multi-byte languages, including Chinese and Japanese.

This international character data can also contain embedded nulls as Tcl keeps track of the string length, not relying on a null termination character. Therefore, the limitation that variables could not contain embedded nulls in previous versions of Tcl no longer exists.

Default action

The default action for data entering a Tcl interpreter through an input channel is to convert it from the external encoding to UTF-8. Data leaving the Tcl interpreter through an output channel undergoes the reverse conversion, that is, it is converted back to the external encoding). This default behavior is changed through Tcl command options. Channel input/output can be configured to perform no encoding conversions or to perform specific conversions.

The system Tcl extensions, and Tcl Xlate fragments, move data into and from Tcl interpreters as arrays of bytes instead of Tcl strings. No encoding conversions are performed when data is moved into an interpreter. Within the interpreter, Tcl commands requiring String data can convert the array of bytes to a UTF-8 string using ISO8859-1 encoding.

When a system Tcl extension moves data from an interpreter, the data is extracted as an array of bytes. If there is not a currently valid byte array representation of the data, then it is regenerated from the String representation. It does this by doing the UTF-8 to 16-bit Unicode encoding conversion and truncating the high-order byte.

Note: This operation works only for ISO8859-1 data.

Perform explicit external encoding to UTF-8 and reverse conversions to avoid data corruption and to perform correct transformations on the Unicode values in the data.

Internationalization encoding functionality

The internationalization encoding functionality is transparent for installations which use only 7-bit ASCII characters within scripts and other data sources. It works correctly without any user coding changes. This is because the UTF-8 representation of a 7-bit ASCII value is identical to the ASCII representation.

Users who have scripts or data sources that contain single-byte values above the ASCII range can modify Tcl processing to be "encoding aware". That is, ordinal character values in the range 128-255. These modifications are to take advantage of Tcl encoding conversion functionality and to prevent unintended default Tcl encoding conversions.

If the input data encoding is not the same as the system encoding, then input/output has the potential to corrupt the data. This happens if the correct encoding is not configured for the channel. This is especially true for default code pages which have unassigned values.

Encodings can be specified for a channel with the Tcl `fconfigure` command. Explicit encoding operations are performed on strings using the `encoding convertfrom` and `encoding convertto` commands. These commands are useful in cases where the default conversion of external data to UTF-8 does not achieve the expected result.

ICU

International Components of Unicode (ICU) is a widely used set of C/C++ and Java libraries providing Unicode and Globalization support for software applications. ICU is widely portable and gives applications the same results on all platforms and between C/C++ and Java software.

Converter alias table

The ICU alias table is available to see how alias names or byte sequences are mapped certain ways.

See: <http://source.icu-project.org/repos/icu/icu/trunk/source/data/mappings/convtrts.txt>.

This table is not meant to be read by newcomers to ICU. This is the main reason why the **Converter Explorer** exists. It does contain comments that some users might find helpful. This alias table can contain information that is more current than your copy of ICU or what is currently available in **Converter Explorer**. The bottom of each **Converter Explorer** page describes the version of ICU it uses.

Converter Explorer

In the Converter Explorer, you can explore the aliases and properties of each ICU converter.

See: <http://www.icu-project.org/icu-bin/convexp>.

Viewing the aliases and standards–ICU Converter Explorer

IANA, Internet Assigned Numbers Authority, is the main source of converter aliases on the Internet. IANA does not specify the Unicode mappings for every codepage and alias, and every platform supports other aliases in addition to the IANA aliases. ICU provides a way to target the codepage conversion based upon the standard or platform. Then you can use the correct converter name and implementation that are based upon which standard you are targeting.

You can change the view of aliases for each standard by selecting the appropriate standard at the top of the page. Then, you can see the subset of aliases that a standard or platform can recognize.

For example, if you select IANA and ALL and select **ShowAliases**, you see all aliases recognized by IANA and ICU. Notice that the IANA set of aliases is a subset of all ICU aliases.

The Internal Converter Name column is also known as a canonical name. The canonical name is a unique ICU converter name. It is usually based upon the UTR #22 naming scheme. The canonical name is always guaranteed to be the correct converter that you require in a particular ICU release. At times, though, the mapping tables get updated between ICU releases, so this converter can change at that time. API functions similar to `ucnv_getCanonicalName()` and `ucnv_getName()` return this value. The `ucnv_getStandardName()` function requires this name as an argument.

The All Aliases column is not a real standard. It is a special way to see all of the aliases for a specific converter regardless of which standards support the converter's alias names.

The Untagged Aliases column is also not a real standard. It is a special way to see all of the aliases that are not associated with any particular standard. An alias in this column indicates it is a name of an alternate mapping table with the same name under a different standard. It can also indicate that this is a rarely used alias and its use is discouraged.

Viewing the converter details

After you have selected a converter to view, click the link to see all of the details about that converter.

This table shows the details that are listed on that page:

Converter detail	Description
Type of converter	This is the internal converter implementation used. The <code>ucnv_getType()</code> API returns this value.
Minimum number of bytes	This is the minimum number of bytes required by this encoding.
Maximum number of bytes	This is the maximum number of bytes required by this encoding.
Substitution character	This is the byte sequence used when a converter encounters an unmappable Unicode character.
Is ASCII [<code>\x20-\x7E</code>] compatible?	<p>This is the byte range <code>\x20</code> to <code>\x7E</code> compatible with ASCII? For example, some codepages map the ASCII backslash <code>\x5C</code> to the Yen Symbol <code>\u00A5</code>. Sometimes, special shift bytes are required to display the ASCII range, and ASCII does not know about codepage shifting or escape sequences. Some codepages are EBCDIC based.</p> <p>Only the range [<code>\x20-\x7E</code>] is used for this comparison. This is because some ISO controls are rotated, and most users are interested in the graphical interpretation of ASCII.</p>
Is ASCII [<code>\u0020-\u007E</code>] ambiguous?	This is the value that <code>ucnv_isAmbiguous()</code> returns. When this value is <code>TRUE</code> , it usually implies that this is a non-ASCII compatible codepage and an ASCII compatible codepage is available.
Contains ambiguous aliases?	This indicates at least one of the aliases for this converter is also on a different converter.
Converters with conflicting aliases	If there are any converters with conflicting aliases, then this has the list of converters with their conflicting alias and standard. Care should be taken when using any of the aliases on this list when a standard is not specified on the ICU conversion API. This information can usually be queried from <code>ucnv_getCanonicalName()</code> or <code>ucnv_getStandardName()</code> .

Converter detail	Description
Always generates Unicode NFC?	<p>When this value is TRUE, a conversion from this codepage to Unicode always generates Unicode in Normalization Form Composed (NFC). When this value is UNKNOWN, there is a possibility that this converter generates Unicode text that is not in NFC depending on the input. It also applies an NFC transformation that could change the original text.</p> <p>This value is derived by creating a Unicode Set with the value <code>[[[:NFC_Quick_Check=yes:]]&[:ccc=0:]]</code>. Then, you can confirm that it is a full superset of the codepage's Unicode Set. More details about Unicode Normalization can be found in Unicode Standard Annex #15.</p>
Contains BiDi characters?	<p>When this value is TRUE, then a conversion to or from this codepage can contain bidirectional characters. These are right to left characters, for example, Hebrew and Arabic characters. When displaying data from this codepage, you might require to apply the BiDi algorithm described in Unicode Standard Annex #9.</p>
List of languages representable by this codepage	<p>This is a list of languages that are representable by this codepage. This data comes from <code>ucnv_getUnicodeSet()</code> and <code>ulocdata_getExemplarSet()</code>, and makes sure that the returned <code>UnicodeSet</code> for the language is a complete subset of the given codepage. The list of languages comes from <code>uloc_getAvailable()</code>.</p>
Set of Unicode characters representable by this codepage	<p>This is a list of Unicode characters that are representable by this codepage. For example, some text data is written as the bytes of the codepage and converted to Unicode. This is the set of possible Unicode characters to which the text could be converted to. This set can contain multi-codepoint characters. This data comes from <code>ucnv_getUnicodeSet()</code>.</p>

Tcl interpreters

Multiple Tcl interpreters are maintained by the system. These are used to run user Tcl scripts or to perform engine-specific processing. As part of these operations, the system moves data into and from Tcl interpreters.

In all cases, the system moves data in binary form across the interface with the Tcl interpreter. Within the Tcl interpreter, if this binary data is processed in a String context, Tcl performs the default encoding conversion from ISO8859-1 to UTF-8. You can precede a string context operation on the data with an explicit encoding

`convertfrom` operation specifying the actual source encoding. If no string operations are performed on the data when it is in the interpreter, then encoding conversion is not performed.

Points to remember

When working with international character data:

- All Tcl scripts must follow Tcl 8.4 conventions.
- The system exchanges binary data with Tcl 8.4 only, that is, channel input/output in Tcl scripts must be binary. This permits the system to work with both "Unicode aware" and "non-Unicode aware" Tcl scripts.
- All existing Tcl 8.0 scripts, for example, non-Unicode aware, can be reused as-is with minor changes. The channel input/output must be in binary.
- You must programmatically convert binary data into Unicode to take advantage of Tcl 8.4's Unicode support.
- Some literal strings might not be valid in Tcl scripts.
- There is automatic support for encoding conversion in XML.
- Non-ASCII characters are not permitted as XML tags.

XML encoding conversion

The system supports automatic conversion of encoding in XML files. In XML-to-XML translations, the system automatically converts from the input encoding to the output encoding, if necessary.

The output encoding is specified by storing a value into the destination `?xml.&encoding` XML path. By default, the input encoding is automatically copied to the output encoding.

Working with non-ASCII XML tags

This section discusses the issue of non-ASCII tags in XML documents. A DTD is used as an example. The same approach can also be applied to XML schema.

This example shows a DTD and XML file with non-ASCII tags. Because the system cannot natively handle this, the solution is to substitute the non-ASCII tags with valid ASCII sequences. The XML file can then be handled.

After translation, ASCII sequences are substituted back to their non-ASCII values.

For example:

dtd: EXâMPLE2.dtd

```
<?xml encoding="ISO8859-1"?>
  <!ELEMENT EXâMPLE2 (â,B,C)+>
<!ELEMENT â (#PCDATA)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```

XML file

```
<EXAMPLE2><â>TOSHâ0</â><B>KABUTO</B><C>rømr</C>
</EXAMPLE2>
```

To perform this substitution:

- 1 Replace the â in the DTD with a two-character ASCII substitute. This involves renaming the file and substituting the non-ASCII characters. You can use _a as a substitute for â. Do the substitution with an editor or use Tcl commands.
- 2 Make your translation using this modified DTD. If you are translating from non-XML into XML, then use a callout at the end of the translate to substitute the message back to non-ASCII.

The CALL operation creates a new message with the _a substituted back to â. The SUPPRESS operation prevents the default message from being generated.

If your translation goes from XML, then you can use a translate pre-procedure to substitute out the non-ASCII tags before translating. You can also put the post-processing into a translate post-procedure. The CALL is best to use because it works with the translate tester.

Using XML TRXID determination

When you use XML TRXID determination, the TRXID is extracted from the XML message before any translate procedures are run. Therefore, the TRXID contains non-ASCII values that cannot be routed. If you are using XML TRXID, then put the procedure to substitute out the non-ASCII characters in inbound TPS processing. By doing so, the TRXID does not contain non-ASCII characters.

Arbitrary precision math operations

`mpexpr` is a Tcl command that evaluates an expression with multiple precision math.

To use this command, a Tcl script or `hcitcl` must use this before processing:

```
package require mpexpr
```

Note: This is only required once per run of the program doing the loading. The extension remains loaded in memory, so that it can be used as many times as required after loading.

`mpexpr` is based on Tcl's built-in `expr` command, and shares many similarities with `expr`. `mpexpr` concatenates args by adding separator spaces between them. It then evaluates the result as a Tcl expression, and returns the value.

The operators that are permitted in Tcl expressions are a subset of the operators permitted in C expressions. They have the same meaning and precedence as the corresponding C operators. Expressions almost always yield numeric results as integer or floating-point values.

For example, the expression `mpexpr 8.2 + 6` evaluates to 14.2.

Tcl expressions differ from C expressions in the way that operands are specified. Tcl expressions support non-numeric operands and string comparisons.

For example:

```
package require mpexpr arg ?arg arg ... ?
mpformat formatString ?arg arg ... ?
global mp_precision
```

The global variable *mp_precision* determines the number of significant digits that are retained during evaluation.

If *mp_precision* is unset, then 17 digits of precision are used.

The maximum value of *mp_precision* is 10000. Larger values require increasingly longer run times. Setting *mp_precision* to an illegal value generates an error.

Operands

A Tcl expression consists of a combination of operands, operators, and parentheses. White space can be used between the operands and operators and parentheses; it is ignored by the expression processor. Where possible, operands are interpreted as integer values.

Integer values can be specified in:

- Decimal. This is the normal case.
- Octal, if the first character of the operand is 0.
- Hexadecimal, if the first two characters of the operand are 0x.

If an operand does not have one of the integer formats given above, then it is treated as a floating-point number, if possible.

Floating-point numbers can be specified in any of the ways that are accepted by an ANSI-compliant C compiler. The f, F, l, and L suffixes are not permitted in most installations.

For example, all of these are valid floating-point numbers: 2.1, 3., 6e4, 7.91e+16.

If no numeric interpretation is possible, then an operand is left as a string. Only a limited set of operators can be applied to it.

Operands can be specified as:

- A numeric value: integer or floating-point.
- A Tcl variable, using standard \$ notation. The variable's value is used as the operand.
- A string that is enclosed in double quotes. The expression parser performs backslash, variable, and command substitutions on the information between the quotes, and use the resulting value as the operand.
- A string that is enclosed in braces. The characters between the open brace and matching close brace are used as the operand without any substitutions.
- A Tcl command that is enclosed in brackets. The command is run and its result is used as the operand.
- A mathematical function whose arguments have any of the above forms for operands, such as `sin($x)`.

Where substitutions happen, for example, inside quoted strings, they are performed by the expression processor. An additional layer of substitution might already have been performed by the command parser

before the expression processor was called. It is usually best to enclose expressions in braces to prevent the command parser from performing substitutions on the contents.

For some examples of expressions, variable *a* has the value 3 and variable *b* has the value 6.

The commands on the left side of each of the lines produce the value on the next line:

```
mpexpr 3.1 + $a
6.1
mpexpr 2 + "$a.$b"
5.6
mpexpr 4*[llength "6 2"]
8
mpexpr {{word one} < "word $a"}
0
```

Message extensions

These commands are used to manipulate message objects in Tcl.

The Tcl extensions in the message category perform operations on hci message objects that can contain large amounts of arbitrary data. Metadata that is associated with each message can reflect and affect how the message proceeds through the engine.

Message objects support data with null characters embedded within them.

Message objects can read and write data directly to and from open file handles and support three types of message boundary determination:

- len10: 10-byte length-encoded
- nl: Newline-terminated
- raw: No externally visible message boundary

In some commands, use offset and length values to specify message regions.

If the offset is not specified, then the beginning of the message is assumed. Offsets beyond the end of the message refer to the character beyond the end of the existing data.

If the length is not specified, then the remainder of the message is used. Lengths that exceed the remainder of the message data encompass the remainder of the message. A length value of end also encompasses the remainder of the message.

Lock message objects to prevent updates or removal. Attempts to modify or remove locked objects generate error conditions. See [Locking message objects](#)

A `-binary` option is available for `msgcreate`, `msgset`, `msginsert`, and `msgappend`. When this is set, the message data is handled as binary data.

Creating message objects using msgcreate

Use the `msgcreate` command to create new messages. It returns the new message object's handle. Use the message object's handle with other MSG extension commands to manipulate the message.

For example:

```
Input: msgcreate
Output: message0
```

- The engine distinguishes between two message classes: engine and protocol.

For example, to set a new message's class with `msgcreate`:

```
Input: msgcreate -meta { CLASS protocol }
Output: message1
```

An equivalent command is:

```
Input: msgcreate -class protocol
Output: message1
```

Both versions of the command set the newly-created message object to be a protocol class message.

By default, `msgcreate` produces engine messages.

- The engine distinguishes between two message types: data and reply.

If specified, then data is the initial value for the new message.

For example:

```
Input: msgcreate -meta { TYPE reply }
Output: message2
```

An equivalent command is:

```
Input: msgcreate -type reply
Output: message2
```

By default, `msgcreate` produces data messages.

- The `msgcreate` command also sets the message's recovery state. `USERECOVERDB` specifies that the message undergoes normal recovery database processing if/when it leaves the interpreter and enters the engine. By default, new messages are TYPE data, indeterminate CLASS, and not handled by the recovery database. Use the `USERECOVERDB 1` flag to permit messages to be saved in the recovery database.

For example:

```
Input: msgcreate -meta { USERECOVERDB 1 }
Output: message3
```

An equivalent command is:

```
Input: msgcreate -recover
Output: message3
```

- Creating a new protocol reply message:

Note: Specify all of the options and flags required when doing this, because after the message is created, these values cannot be changed.

For example:

```
Input: msgcreate -class protocol -recover -type\ reply
Output: message4
```

An equivalent command is:

```
Input: msgcreate -meta { CLASS protocol\ USERECOVERDB 1 TYPE reply }
Output: message4
```

Use the "--" flag to separate the options from the rest of the args.

For example:

```
Input: msgcreate -- -meta
Output: message0
```

By default, message objects are created without data. Specify the initial value after the options and flags.

- Creating a new protocol reply message initialized with data:

For example, to create a new protocol reply message that is recoverable and initialized with data:

```
Input: msgcreate -class protocol -recover -type\ reply "some data"
Output: message5
```

An equivalent command is:

```
Input: msgcreate -meta {CLASS protocol\ USERECOVERDB 1 TYPE reply} "some data"
Output: message5
```

Message objects can hold up to 4,294,967,295 (that is, $2^{32} - 1$) characters. An error happens if an attempt is made to extend a message beyond this limit.

The `msglength` command returns the number of characters in a message.

For example:

```
Input: msglength message6
Output: 24
```

- Information on most extensions is available from the `hcl` window. You can use the keyword `help` followed by the command that needs clarification. Additionally, command syntax is available in the `hcl` window by entering the command with incorrect syntax.

An example of using `msgcreate` is:

```
Input: help msgcreate
Output: msgcreate -meta ?CLASS? ?TYPE? USERECOVERDB ?data?
```

The available options for `msgcreate` are also displayed if the command is run with an `-h` option. This option usually indicates help.

For example:

```
Input: msgcreate -h
Output: Error: bad option "-h": should be -class,
        -meta, -recover, -type, or --
```

Destroying message objects

The `msgdestroy` command processes any number of message handles and returns an empty string. Destroy created messages at any time. Destroying a message removes its handle from the interpreter and frees any resources it is consuming. It also removes the message from the recovery database.

For example:

```
Input: msgdestroy message0
Output:

Input: msgdestroy message1 message2
Output:
```

Locking message objects

Place locks on message objects to prevent certain actions. For example, the engine might pass a message into a UPoC, prohibiting the user from destroying, or possibly even modifying the message.

Query and set locks with the `msglock` command. This command returns a list of locks applied to the message. If new lock flags are specified, then the return includes the new locks.

- The `-nodestroy` flag renders the message undestroyable.
- `-nowrite` prevents any one from updating the message or its metadata.

For example:

```
Input: msglock message3
Output:

Input: msglock -nodestroy message3
Output: nodestroy

Input: msglock -nowrite message3
Output: nodestroy nowrite

Input: msglock message3
Output: nodestroy nowrite
```

An error happens if an attempt is made to destroy or modify a locked message. After a lock is applied to a message, it cannot be removed.

Note: These locks apply only to Tcl commands; they have no effect on engine behavior.

Listing message handles

The `msglist` command returns the list of message handles in the interpreter, in no particular order.

For example:

```
Input: msglist
Output: message3 message4 message5 message6
```

Querying message data

The primary method of retrieving data from a message is the `msgget` command. By default, it returns the entire contents of the message.

For example:

```
Input: msgget message6
Output: some protocol reply data
```

Specify an offset to skip that many characters.

For example:

```
Input: msgget message6 5
Output: protocol reply data

Input: msgget message6 5 end
Output: protocol reply data
```

Use a length value to limit the retrieval.

For example:

```
Input: msgget message6 5 8
Output: protocol
```

Null characters

Message objects handle null characters internally.

For example, `message7` contains: This data has a <NULL> embedded NULL. NULL is the null character. You can use the `-cvtnull` option with `msgget` to replace that null character with a more readable replacement.

```
Input: msgcreate -class protocol -recover -type reply
Output: message7

Input: msginsert -cvtnull X message7 "this data has an X embedded NULL"
Output:

Input: msgget message7
Output: this data has an Ç embedded NULL

Input: msgget -cvtnull Z message7
Output: this data has an Z embedded NULL
```


Use any replacement character with the `-cvtnull` option.

Modifying message data

You can modify message data by adding data to a message or using the null option:

- Adding data to a message with `msginsert`:

`msginsert` adds a string to the message. An insertion offset can be specified. If it is not, then it defaults to 0, and the data are prepended to the message.

For example:

```
Input: msginsert message6 "This message contains "
Output:

Input: msgget message6
Output: This message contains some protocol reply data

Input: msginsert message6 "more " 27
Output:

Input: msgget message6
Output: This message contains some more protocol reply dat
```

- Null option:

`msginsert`'s `-cvtnull` option is the opposite of `msgget`'s. Use it to specify which character to convert to null as the string is added to the message.

For example:

```
Input: msgcreate -class protocol -recover -type reply
Output: message4

Input: msginsert -cvtnull X message4 "this data has an X\ embedded NULL"
Output:

Input: msgget message4
Output: this data has an Ç embedded NULL

Input: msgget -cvtnull Z message4
Output: this data has an Z embedded NULL
```

This command returns an empty string.

Adding data to a message with `msgappend`

This command adds data to the end of a message. It accepts a `-cvtnull` option such as `msginsert`.

For example:

```
Input: msgappend message5 ", for example"

Input: msgget message5
Output: some data, for example

Input: msgappend -cvtnull _ message5 ", with_a NULL"
```

```
Input: msgget message5
Output: some data, for example, with

Input: msgget -cvtnull X message5
Output: some data, for example, withXa NULL
```

Modifying data with msgset

You can modify data with `msgset` using these methods:

- `msgset` is the most general way to modify message data. Its offset and length parameters specify the region of the message to replace with new data. If neither is specified, then the entire message is replaced with the new data.

For example:

```
nput: msgset message6 "This is message6's new\ value"

Input: msgget message6
Output: This is message6's new VALUE
```

- Replacing a 0-length region is an insertion. If the offset is at or beyond the message's length, then it is an append.

For example:

```
Input: msgset message6 "string " 5 0

Input: msgget message6
Output: This string is message6's new value
```

- `msgset` can replace some portion of the message data with a new string. If the new and old strings have different lengths, then `msgset`

For example: adjusts the message length.

```
Input: msgset message6 "These characters are" 2 12

Input: msgget message6
Output: These characters are message6's new value
```

- Replacing a region with an empty string removes that region.
adjusts the messageFor example:

```
Input: msgset message6 {} 31 4

Input: msgget message6
Output: These characters are message6's value
```

- `msgset` processes the `-cvtnull` option in the appropriate manner.

For example:

```
Input: msgset -cvtnull - message6 "Some-data"

Input: msglength message6
```

```
Output: 9
Input: msgget message6
Output: some
Input: msgget -cvtnull Z message6
Output: someZdata
```

Message file access: Reading message data

Modify message contents by reading from open file handles. Reading data from a file directly into a message object preserves any null characters embedded in the data. The file-read commands neither have nor require the `-cvtnull` option.

Message objects support three styles of reading:

- `nl` (newline-terminated) reads up to, discards, the next newline character. This style is identical to that supported by the File protocol driver.

Use this style to assign a line of input text to a message.

For example:

```
Input: msgread nl message6 stdin\
This is user input
Output: 18
Input: msgget message6
Output: This is user input
```

- `len10` (10-byte length-encoded) treats the next 10 characters in the file as a 10-digit ASCII representation of the number of characters in the message itself. This style is identical to that supported by the File protocol driver.
- `raw` reads the user-specified number of characters to read.

When using this style, assume direct control of the number of bytes read.

Monitor file reads more closely with the `-stats` option.

When the `-stats` option is used and a variable name is supplied as its argument, the variable is assigned a list of two elements: `todo done`.

- `todo` is 0 for `nl` read, the number of bytes in the `len10` header, or the number of bytes requested in a `raw` read.
- `done` is the actual number of bytes read.

Comparing these values after a failed read can give some insight into the nature of the failure.

For example:

```
Input: open /etc/passwd r
Output: file4
Input: msgread -stats s nl message6 file4
Output: 22
Input: echo $s
Output: 0 22
Input: msgread -stats s raw message6 file4 6
Output: 6
```

```
Input: msgget message6
Output: daemon
```

```
Input: echo $s
Output: 6 6
```

No matter which style is used, these commands return the number of bytes read.

All of the file-reading commands operate on open file handles.

This table shows the file-read commands and their comparable message modification commands:

File-read command	Message modification command
msgappread	msgappend
msginsread	msginsert
msgread	msgset

Message file access: Writing message data

Write message data directly to open file handles. Similar to reading directly from files, this bypasses the normal Tcl interface and preserves null characters.

Similar to `msgget`, `msgwrite` defaults to the entire message contents, but offset and length values can be specified to limit the data written.

Through the `msgwrite` command, the same style choices are available as reading:

- The `nl` style appends a newline character after the data.

For example:

```
Input: msgwrite nl message6 stdout
Output: daemon
```

- The `len10` style prepends a 10-character length-encoding string before the data.

For example:

```
Input: msgwrite len10 message6 stdout 2
Output: 0000000004emon
```

A prompt is given after the data because there is not a trailing newline character.

- The `raw` style writes only the data.

For example:

```
Input: msgwrite raw message6 stdout 3 2
Output: mo
```

A prompt is given after the data because there is not a trailing newline character.

Metadata

System messages have extra information associated with them called metadata. Each message has many metadata fields. These fields have a unique purpose in recording where the message has been, where it is going, or how to process it.

The easiest way to see all of a message's metadata is the `msgdump` command.

For example:

```
Input: msgdump message6
```

Note:

Access a message's metadata with the `msgmetaget` and `msgmetaset` commands. These commands present a keyed-list interface. In this case, each metadata field has a unique key which is used to retrieve or modify that field. The last entry, labeled "message," is a raw dump of the message data. ASCII hexadecimal values are used in lieu of non-printable characters. The rest of the entries are metadata fields.

Similar to `keylget`, `msgmetaset` returns the list of available keys.

For example:

```
Input: msgmetaget message6
```

```
Output: CLASS DATAFMT DESTCONN DRIVERCTL FLAGS ISSTATICDIRTY ISVARDIRTY MID ORIGDESTCONN ORIG
SOURCECONN PRIORITY RETRIES
SKIPXLT GROUPID GROUPMID SOURCECONN SOURCEMID SOURCEMSIDX STATE TIMEIN TIMEOUT TIMEQCUR TIMEQTOT
TIMEXLT TYPE USERDATA
USERRECOVERDB XLTTHREAD
```

Managing metadata fields

These flags are used in managing metadata fields:

- `-ro` and `-rw` flags

Some of the metadata fields are managed exclusively by the engine, and cannot be changed. These fields are read-only (`-ro`). Modifiable fields are called read-write (`-rw`). Use these flags to display only the read-only or read-write keys, respectively.

For example:

```
Input: msgmetaget -ro message6
```

```
Output: CLASS ISSTATICDIRTY ISVARDIRTY MID ORIGDESTCONN ORIGSOURCECONN RETRIES GROUPMID
SOURCEMID
SOURCEMSIDX STATE TIMEIN TIMEOUT TIMEQCUR TIMEQTOT TIMEXLT TYPE USERRECOVERDB XLTTHREAD
```

```
Input: msgmetaget -rw message6
```

```
Output: DATAFMT DESTCONN DRIVERCTL FLAGS PRIORITY SKIPXLT GROUPID SEPCHARS SOURCECONN USERDATA
```

- `-all` flag

This flag displays all of the keys, and is the same as using none of the flags.

For example:

```
Input: msgmetaget -all message6
Output: CLASS DATAFMT DESTCONN DRIVERCTL FLAGS ISSTATICDIRTY ISVARDIRTY MID ORIGDESTCONN
ORIGSOURCECONN
PRIORITY RETRIES SKIPXLT GROUPID GROUPMID SEPCHARS SOURCECONN SOURCEMID SOURCEMSIIDX STATE
TIMEIN TIMEOUT
TIMEQCUR TIMEQTOT TIMEXLT TYPE USERDATA USERECOVERDB XLTHREAD
```

Specifying a key

Use `CLASS` or `TYPE` to specify a particular key to retrieve that metadata value.

For example:

```
Input: msgmetaget message6 CLASS
Output: protocol
Input: msgmetaget message6 TYPE
Output: reply
```

Modifying read-write fields

Use `msgmetaset` to modify read-write fields. Similar to `keylset`, specify any number of key-value pairs.

For example:

```
Input: msgmetaset message6 PRIORITY 5122 DESTCONN some_name
Input: msgdump message6
```

This command returns an empty string.

The FLAGS metadata field

The `FLAGS` metadata field is a collection of individual boolean flags, each of which has a unique name. Similar to metadata fields, some flags can be modified, but not others.

Some users require a binary mapping of bit values to message flags. The listed message flags are the externally visible bit values that are visible through Tcl.

For example, this is useful in writing a Tcl script to convert old format SMAT files without metadata to the new style that has metadata. `msgmetaset` does not work because there is no message handle. Knowing the flag names does not work.

Therefore, the list of Tcl flag names describes the binary position of each external message flag, so that you can construct the `FLAGS` field.

External message flags bit masks:

- `should_be_freed`

Used by internal memory management.

Bit Value: 0x00000001

Tcl Writable: FALSE

- `icl_owns_data`

Whether the local copy of the message can be destroyed if sent to a thread in another process.

Bit Value: 0x00000002

Tcl Writable: FALSE

- `expect_reply`

Automatically set when the thread is configured to await replies. When set, the threads await a reply after writing the message through the protocol connection.

Bit Value: 0x00000004

Tcl Writable: TRUE

- `is_forwarded`

Set if the message is ever forwarded from one thread to another.

Bit Value: 0x00000008

Tcl Writable: FALSE

- `is_enqueued`

TRUE when MSG is on a queue; otherwise, FALSE.

Bit Value: 0x00000010

Tcl Writable: FALSE

- `last_in_group`

Set to indicate that this message is the last in a related group.

Bit Value: 0x00000020

Tcl Writable: TRUE

- `proto_timeout`

Set by the protocol driver to describe a protocol-send failure due to time-out. Intended for use by SENDFAIL TPS procedures.

Bit Value: 0x00000040

Tcl Writable: TRUE

- `proto_nak`

Set by the protocol driver to describe a protocol-send failure due to nak. Intended for use by SENDFAIL TPS procedures.

Bit Value: 0x00000080

Tcl Writable: TRUE

- `is_resent`

Set if this message was resent back into the engine.

Bit Value: 0x00000200

Tcl Writable: TRUE

- `recovered`

Recovers the message from the database.

Bit Value: 0x00000800

Tcl Writable: TRUE

- `is_on_disk`

Bit Value: 0x00001000

Tcl Writable: FALSE

- `keep_on_disk`

Bit Value: 0x00002000

Tcl Writable: FALSE

- `use_rdb`

Set to indicate that this message goes to the recovery database if there is an engine failure.

Bit Value: 0x00008000

Tcl Writable: FALSE

- `prewrite_done`

Set if this message has a prewrite procedure.

Bit Value: 0x00010000

Tcl Writable: FALSE

`msgdump` displays the flags as a hexadecimal value. When the `FLAGS` metadata field is retrieved, `msgmetaget` returns the list of enabled flags.

Changing the FLAGS field

To change the `FLAGS` field, prepare a keyed list where the keys are the individual flags and each key's value is that flag's boolean value. Legal boolean values are 0, 1, on, off, true, and false.

For example:

```
Input: msgmetaset message6 FLAGS { <some flags\ keyed list...> }
Input: msgmetaset message6 FLAGS { {expect_reply\ TRUE} {proto_nak 0} }
```

Limitations and guidelines

There are other limitations and guidelines for metadata fields:

- Thread names that are used with the `SOURCECONN` and `DESTCONN` metadata fields cannot be longer than 64 characters.
- The `DESTCONN` metadata field can accept any number of legal thread names. The `SOURCECONN` field accepts only one.
- Legal `PRIORITY` values that can be set are between 4096 and 8192, inclusive.
- The `USERDATA` field is not used by the system at all. Use this field for anything you require. For example, passing control data between TPS procedures.

Duplicating a message

The `msgcopy` command creates a new message as a duplicate of another, with the copy having a unique message ID value. All but one of the metadata fields are copied.

For example:

```
Input: msgcopy message6
Output: message2
```

Message transliteration

Transliterating is the process of replacing all instances of characters in one class with corresponding characters in another class. A common example is to convert a message from the EBCDIC character set into ASCII. In this case, every instance of the EBCDIC 'A' character (0xc1) becomes ASCII 'A' (0x41); EBCDIC 'B' (0xc2) becomes ASCII 'B' (0x42), and so on.

A character map table describes a transliteration from one character set to another. It is represented by a list of 256 values. The input character value serves as the index into the list. The value at that position is the transliterated value. In the preceding example, the (0xc1)th (193) entry would be 0x41 and the 0xc2th (194) would be 0x42.

Each character-map table:

- Describes a one-way mapping. Any number of input characters (indices) can map to the same value.
- Has a unique name. The `adddatamap` command associates a name with a new table.

For readability, the elements are in lines of eight hex characters. The lines are arranged in pairs to make 16 sets of 16 characters. To find the 0xc1-th element, count down c pairs, starting with 0. Then, count over 1 element, starting with 0.

Using msgmapdata

This command uses a character-map table to transliterate a message.

For example:

```
Input: msgcreate "This is ASCII"
Output: message1

Input: msgmapdata message1 ibm_a2e fi

Input: msgdump message1
Output: [...] message : '\xe3\x88\x89\xa2@\x89\xa2@\xc1\xe2\xc3\xc9\xc9'

Input: msgmapdata message1 ibm_e2a fi

Input: msgdump message1
Output: [...] message : 'This is ASCII'
```

This command returns an empty string.

Using tbllookup

Use this command to reference a system lookup table from Tcl:

```
tbllookup ?<side>? <table> <value>
```

- *side* is the input side to use.
- *table* is the name of the table to use.
- *value* is the input value to use.

This maps a value through a lookup table, and returns the table's output value. An error happens if the table cannot be found.

Message references

The message is the fundamental object in the system. The message object manages both the raw data that makes up the message and the additional information, metadata, about the message.

adddatamap

```
adddatamap mapName mapList
```

`mapList` is a list of 256 elements: one for each ASCII value. Each element is the ordinal value of the character to which the original character should be mapped when using the table. After a table is created, a message is mapped using the `msgmapdata` command.

This command returns an empty string.

msgappend

```
msgappend ?-cvtnull char? msgId data
```

This command appends the specified data to `msgId`. If the `-cvtnull` option is used, then every instance of `char` is converted to a null character before the data is appended to the message.

This command returns an empty string.

```
msgappread ?-stats statsVar? len10 msgId fileId
msgappread ?-stats statsVar? nl msgId fileId
msgappread ?-stats statsVar? raw msgId fileId len
```

This command reads the next message from `fileId`, an open file handle. It then appends the data to `msgId`.

If a `len10` or `nl` read is specified, then the appropriate message boundary is used to terminate the read.

Supply a read length, `len`, for `raw` reads.

If a `statsVar` variable name is supplied, then the variable is set to contain a list of two values: `todo` `done`.

- `todo` is the number of bytes requested. This is the length implied by a `len10` record header: 0 for `n1` reads, or the `len` value for `raw` reads.
- `done` is the actual number of characters read.

These values can be useful when a read fails.

This command returns the number of bytes read.

msgcopy

```
msgcopy msgId
```

This command produces a new message with data identical to `msgId`.

Some metadata fields are copied, some are not. For example, a new message receives a different `mid` and its `srcmid` is set to the original message's `mid`.

This command returns the new message handle.

msgcreate

```
msgcreate -meta ?CLASS? ?TYPE? USERECOVERDB ?data?
```

This command creates a new message.

- `CLASS` specifies an hci message class: `engine` or `protocol`.
- `TYPE` specifies an hci message type: `data` or `reply`.
- `USERECOVERDB` specifies that the message undergoes normal recovery database processing when it leaves the interpreter and enters the engine.

By default, new messages are `TYPE data`, `CLASS engine`, and not handled by the recovery database.

If specified, then `data` is the initial data value for the new message.

This command returns the new message handle.

msgdestroy

```
msgdestroy msgId ?msgId...?
```

This command destroys the specified messages.

This command returns an empty string.

msgdump

```
msgdump msgId
```

This prints internal details about the message to `stdout`.

This command returns an empty string.

To dump message metadata into the log except message data, use this command:

```
msgdump mh -nodumpdata
```

msgerror

```
msgerror context msgId ?msgId...?
```

This transitions the given messages to the error database and attaches the context string to it. You can put messages into the error database when not in a TPS environment. For example, when the error disposition cannot be used.

Unlike the error disposition, this command lets the user attach a context/description string to the message.

The command fails if a message is destroy-locked.

This command returns an empty string.

msgfindchar

```
msgfindchar msgID char ?offset? ?length?
```

This command finds the first instance of `char` in `msgId`'s region.

- If `offset` is not specified, then 0 is assumed.
- If `length` is not specified or specified as `end`, then the remainder of the message is used.
- If the character is in the message's data, then the command returns the offset from the start of the region.
- If the character is not in the region, then the command returns -1.

Use `{}` to search for the null character.

msgfinduchar

```
msgfinduchar msgId char ?offset? ?length?
```

This command finds the lowest character greater than or equal to `char`, and is not used in the message region.

- `offset` defaults to 0.

- `length` defaults to the rest of the message. This can also be `end`.

msgget

```
msgget ?-cvtnull char? msgId ?offset? ?length?
```

This command retrieves message data that is from the region specified by `offset` and `length`.

Nulls in the retrieved data are converted to `char` if `-cvtnull` is used.

This command returns the retrieved, converted data.

msggetdat

```
msggetdat ?-cvtnull char? ?-type type? msgId ?offset? ?length?
```

This command returns message data from the region that is specified by `offset` and `length` in a datum object.

Nulls in the retrieved data are converted to `char` if `-cvtnull` is used.

The datum object is type `ch` unless another type value is specified.

This command returns the new datum object's handle.

msginsert

```
msginsert ?-cvtnull char? msgId data ?offset?
```

This command inserts data into `msgId` at `offset`. The default is zero. Instances of `char` in `data` are converted to null if `-cvtnull` is used.

This command returns an empty string.

msginsread

This command reads the next message from `fileId`, an open file handle. It then inserts the data into `msgId` at `offset` of zero, or `offset`, if specified.

```
msginsread ?-stats statsVar? len10 msgId fileId ?offset?  
msginsread ?-stats statsVar? nl msgId fileId ?offset?  
msginsread ?-stats statsVar? raw msgId fileId len ?offset?
```

- If a `len10` or `nl` read is specified, then the appropriate message boundary is used to terminate the read.
- Supply a read length, `len`, for raw reads.

If a `statsVar` variable name is supplied, then the variable is set to contain a list of two values: `todo` `done`.

- `todo` is the number of bytes requested. This is the length implied by a `len10` record header: 0 for `nl` reads, or the `len` value for raw reads.
- `done` is the actual number of characters read.

These values are useful if a read fails.

This command returns the number of bytes read.

msglength

```
msglength msgId
```

This command returns the length of `msgId`'s data.

msglist

```
msglist
```

This command returns a list of valid message handles currently available in the Tcl interpreter.

msglock

```
msglock ?-nodestroy? ?-nowrite? msgId
```

This command modifies and queries a message object's locks. Each flag specifies a lock to set on the message in the interpreter. After set, a lock cannot be removed.

`msglock` returns a list describing the message's lock-state after the new locks, if any, are applied.

The list contains zero or more of these elements:

- `nodestroy`
Message cannot be destroyed
- `nowrite`
Message cannot be modified

msgmapdata

```
msgmapdata msgId mactable
```

This command applies the character conversion described by `maptable` to `msgId`. Map tables are created using the `adddatamap` command.

This command returns an empty string.

msgmetaget

```
msgmetaget ?-rw | -ro | -all? msgId ?key?
```

If a `key` value is not specified, then the available metadata keys are returned.

If a `key` value is specified, then the metadata that is associated with that key is returned.

`-rw` restricts the key list to read-write (modifiable).

`-ro` restricts the key list to read-only.

msgmetaset

```
msgmetaset msgId key value ?key2 value2 ...?
```

This command sets values for one or more metadata keys.

This command returns an empty string.

msgread

```
msgread ?-stats statsVar? len10 msgId fileId ?off?\ ?len?  
msgread ?-stats statsVar? nl msgId fileId ?off?\ ?len?  
msgread ?-stats statsVar? raw msgId fileId len\ ?off?\ ?len?
```

This command reads the next message from `fileId`, an open file handle. It then replaces a part of `msgId` with the data read.

The replacement region begins at offset `off`, 0 if not specified. It includes `len` bytes. These are the remainder of the message data, if not specified.

- If a `len10` or `nl` read is specified, then the appropriate message boundary is used to terminate the read.
- Supply a read length, `len`, for raw reads.

If a `statsVar` variable name is supplied, then the variable is set to contain a list of two values: `todo` `done`

- `todo` is the number of bytes requested. This is the length implied by a `len10` record header: 0 for `nl` reads, or the `len` value for raw reads.
- `done` is the actual number of characters read.

These values can be useful when a read fails.

This command returns the number of bytes read.

msgrouteget

```
msgrouteget msgId
```

This command returns `msgId`'s translation and routing configuration list. Use this command in generate route procedures.

msgrouteset

```
msgrouteset msgId list
```

This replaces `msgId`'s translation and routing configuration list with `list`. Use this command in generate route procedures.

msgset

```
msgset ?-cvtnull char? msgId data ?offset? ?length?
```

This command replaces a portion of `msgId`'s data with `data`. Instances of `char` are converted to null if `-cvtnull` is used. The replacement region begins at `offset`, 0 if not specified, and includes `length` bytes. This is the remainder of the message, if not specified.

msgstats

```
msgstats msgId
```

This command returns a keyed list containing statistics on message memory use.

msgwrite

```
msgwrite style msgId fileId ?offset? ?length?
```

This command writes data from `msgId` to file handle `fileId` using a record style. The region that is written begins at `offset`, or 0 if not specified, and includes `length` bytes. The remainder of the message, if this is not specified.

This command returns an empty string.

MSI extensions and references

Engine Monitor Statistics Interface (MSI) provides access to statistics describing the run-time state of the system. Through this interface, you can determine the run-time state of system processes and threads.

These commands give direct access to the raw information that tools such as `hcinetmonitor`, `hciproccstatus`, and `hciconnstatus` use.

Note: These commands and the data they return are primarily intended for internal use. The interfaces or the data can change at any time.

MSI initialization

Before using any other `msi` command, use the `msiAttach` command to initialize the subsystem. This command makes the MSI data available to every Tcl interpreter in a process.

The first invocation within a process returns an empty string. Each subsequent invocation returns an error stating the subsystem is already initialized.

For example:

```
Input: msiAttach
Input: msiAttach
Output: Error: shared memory region is already attached
```

Querying the MSI table of contents

MSI's Table of Contents (ToC) lists some basic information about the data available.

With no arguments, `msiTocEntry` returns the names of the thread entries in the table of contents.

This syntax is the same as `keylget`.

For example:

```
Input: msiTocEntry
Output: conn_0 conn_1 conn_2 conn_3
```

If a particular thread name is specified, then the command returns that thread's entry. The data are returned in a keyed list.

For example:

```
Input: msiTocEntry conn_0
Output: {INDEX 1} {ALIVE 0} {CREATED 1}
{NAME conn_0}
```

Storing ToC data into a variable

Use `msiTocEntry` to store the ToC data into a variable.

- If the thread has a ToC entry, then `msiTocEntry` stores the data into the variable and returns "1".
- If the thread does not have a ToC entry, then `msiTocEntry` returns 0 and leaves the variable unmodified.

For example:

```
Input: msiTocEntry conn_0 var
Output: 1

Input: echo $var
Output: {INDEX 1} {ALIVE 0} {CREATED 1} {NAME conn_0}
```

Testing the presence of a thread's TOC

You can use `msiTocEntry` to test for the presence of a thread's ToC entry by specifying {}, an empty string, as the variable name.

For example:

```
Input: msiTocEntry conn_0 {}
Output: 1

Input: msiTocEntry some_thread_name {}
Output: 0
```

Querying thread statistics from MSI

Given a thread name, or an INDEX value from `msiTocEntry`, use `msiGetStatSample` to get a statistics sample for a particular thread.

For example:

```
Input: msiGetStatSample conn_0
```

Modifying a variable name

Specify a variable name for `msiGetStatSample` to modify.

Similar to `msiTocEntry`, `msiGetStatSample` returns "0" or "1". Unlike `msiTocEntry`, {}

For example:

```
Input: msiGetStatSample conn_0 var
Output: 1

Input: msiTocEntry some_thread_name var
Output: 0 cannot be
       used.
```

msiAttach

This command attaches the MSI memory region. Use this command for the other MSI extensions to work.

The first time this command is run in an interpreter, it returns an empty string. An error happens every time thereafter.

msiTocEntry

```
msiTocEntry ?threadName? ?retvar | {}?
```

This command is similar to `keylget` for `retvar` handling.

- If `retvar` is not specified, then the value is the keyed list of ToC entry data. If the thread is not found within the ToC, then an error results.
- If `retvar` is specified and the thread is in the ToC, then the keyed list of data is placed in `retvar`. The command returns "1". If the thread is not in the ToC, then the command returns 0 and `retvar` is left unchanged.
- If `retvar` is specified as {}, then the keyed list is not returned, permitting the programmer to determine the presence of the ToC entry.
- If `threadName` is omitted, then the list of threads in the ToC is returned.

msiGetStatSample

```
msiGetStatSample {threadName | index} ?retvar?
```

This command is similar to `keylget` for `retvar` handling.

Specify the thread from which to extract data by name or by index. If the argument is numeric, then it is assumed to be an index. If not, then it is assumed to be a thread name.

- If `retvar` is not specified, then the value returned is the keyed list of static data. If the thread is not found within the ToC, then an error results.
- If `retvar` is specified and the thread is in the ToC, then the keyed list of data is placed in `retvar`. The command returns "1". If the thread is not in the ToC, then the command returns 0 and `retvar` is left unchanged.
- If the specified thread exists but has never been initialized, then an empty keyed list results.

XPM extensions and references

The Xlation Pseudo Machine (XPM) is the heart of the record translation system of the system. XPM's actions are directed by the translation files that are configured with `hcitranslateconfig`.

The Tcl extensions in the XPM category permit access to XPM messages and data during Tcl callouts when XLT is running. XPM handles exist only during these callouts. There is no way to create or destroy them.

Most of XPM's actions are operations, such as copy, table lookup, and math operation. At times, XPM's basic operations might not be sufficient for your needs. Certain problems might require you to use custom Tcl code within an XLT file.

The `xpm` extensions give access to the XPM state when the translation is running.

A working knowledge of these is required:

- Basic XPM concepts such as pre- and post-processor callouts, CALL action code bodies, temporary variables, and error handling.
- Message and datum objects and the commands to manipulate them. See Record formats and processing rules, field naming and addressing. [Datum extensions](#) and [GRM extensions](#).

Architecture

XPM performs record translations by processing field data from one record into another. XPM uses the GRM module to manage access to the input and output records. See [GRM extensions](#).

XPM also manages temporary variables, for example, `@name`, lookup tables, for use with the TABLE command, and an output message prototype.

The output message prototype is the metadata pattern for messages that are produced by the XPM. When messages are produced by the XPM, the new messages inherit many of the prototype's metadata values. For example, messages that are produced by the CONTINUE or SEND actions. These values include destination, priority, and source MID.

Handles

All of the `xpm` commands described in this section operate on an XPM handle. Unlike other system extension objects, XPM handles are not explicitly created.

Instead, handles are included in your Tcl code within callouts from XPM. For example, action pre- and post-processors and CALL action bodies in the `xlateId` variable.

Because XPM handles are created in Tcl callouts from XPM, `xpm` commands are not run interactively. The examples in this section show the command run as if it were interactive.

Querying field data

The `xpmfetch` command is to XPM handles what `grmfetch` is to GRM handles.

This command has access to these sources of data:

- XPM's input record
- XPM's output record

- Temporary variables

Similar to `grmfetch`, `xpmfetch` uses an address string to identify the field data to retrieve. `xpmfetch` addresses are identical to the addresses that are used in XPM action input lists and are:

- `@name`: Temporary field name
- `~name`: Output field name
- `name`: Input field name

Similar to `grmfetch`, `xpmfetch` returns one DAT handle per retrieved item and supports a `-warn` option.

For example, an XPM is running a translation which maps FRL data to HL7.

- To retrieve a particular input subfield:

```
Input: xpmfetch -warn w $xlateId {NAME.[1]}
Output: datum0
```

- To retrieve data already stored in the output:

```
Input: xpmfetch -warn w $xlateId ~0(0).PID(0).00041
Output: datum1 datum2 datum3
```

- To retrieve a temporary variable:

```
Input: xpmfetch $xlateId @temp
Output: datum4
```

Modifying field data

Modify output record fields and temporary fields through the `xpmstore` command. Input record fields cannot be changed.

To change an output record field, use the field name. Use an `@` to access a temporary field. Except for handling temporary fields and operating on an XPM handle, `xpmstore` accepts the same arguments and responds similar to `grmstore`.

For example:

```
Input: xpmstore $xlateId @tempdate c 19950704\ -type dt
Input: xpmstore -list $xlateId 0(0).PID(0).00041\ c "First -d datum0 -v null"
```

Metadata access

Use `xpmmetaget` and `xpmmetaset` to query and modify the XPM output message prototype's metadata. Their operation is identical to `msgmetaget` and `msgmetaset`, with the exception that they use an XPM handle instead of a message handle.

When XPM metadata is modified, the metadata that is inherited by all messages then produced by the XPM is modified.

For example, the `DESTCONN` metadata field is initialized to the route detail's destination list. By default, all of the messages that are produced by that detail are routed to the threads in that list.

Re-route that route detail's messages by modifying the `DESTCONN` metadata value.

For example:

```
Input: xpmmetaget $xlateId
Output: <metadata key list, similar to msgmetaget>
Input: xpmmetaset $xlateId DESTCONN newthread
```

Note: Metadata changes only effect messages that are produced after the change is made. Messages already in XPM's output list are not affected.

Message encoding

Use the `xpmencode` command to encode the data in XPM's output record into a new message object. Similar to `grmencode`, `xpmencode` can produce a warning list with the `-warn` option.

Additionally, the messages produced inherit their metadata values from XPM's output message prototype, instead of a GRM handle.

For example:

```
Input: xpmencode $xlateId
Output: message3
```

This command returns the new message handle.

Appending messages to XPM's output list

XPM produces a list of messages, each with a TPS disposition value. `xpmdispose` appends one or more messages to that output list. All of the messages appended at one time share the same disposition.

The disposition value is one of:

- `continue`
- `error`
- `kill (stop)`
- `over`
- `proto`
- `send`

After processed by `xpmdispose`, message handles are removed from the interpreter.

How the system processes the messages depends upon their metadata values. See [Message extensions](#).

XPM error generation

The `xpmerror` command generates an error condition in the XPM. Specify the severity when an XPM error is generated.

Error conditions are:

- `action`
Error is handled by the XLT statement error control value.
- `curdetail`
Error out only the current translation routing detail but continue the remaining ones.
- `alldetail`
Error out all of the message's translation routing details.

The last argument is a textual description of the error. If the original message that is passed to XPM is transitioned into the error database, then this text message is attached to the message. This can help determine why the translation failed.

To prevent further Tcl processing, `xpmerror` also generates a Tcl error event.

Use the `catch` command to prevent the Tcl event from stopping.

The XPM error cannot be stopped.

For example:

```
Input: xpmerror $xlateId action "action error\ message"
Output: Error: action error message

Input: xpmerror $xlateId curdetail "curdetail\ error message"
Output: Error: curdetail error message

Input: xpmerror $xlateId alldetail "alldetail\ error message"
Output: Error: alldetail error message
```

xpmdispose

```
xpmdispose xpmId disp msgId ?...?
```

This command places one or more messages onto `xpmId`'s output list with disposition `disp`.

`disp` must be one of these TPS disposition values:

- `continue`
- `error`
- `kill (stop)`
- `over`
- `proto`
- `send`

Messages that are passed to XPM by this command are no longer available in the interpreter.

This command returns an empty string.

xpmencode

```
xpmencode ?-warn var? xpmId
```

This encodes a message according to the XPM output data state, and returns its handle.

- Any warnings that are generated during the encoding process are appended to the variable *var*.
- Encoding a message does not affect the translation process.
- Messages that are produced by this command exist only in the interpreter. The user is responsible for destroying them, or passing them to the engine by `xpmdispose`.

xpmerror

```
xpmerror xpmId severity message
```

This generates an XPM error condition.

`message` is a textual description of the error. If the source message is moved to the error database, then `message` is attached to the record as its error context.

`severity` can be any of these values:

- `action`: The error is handled by an XLT statement error control value.
- `curdetail`: Error out only the current translation routing detail, but continue the remaining ones.
- `alldetail`: Error out all of the message's translation routing details

This command also generates a Tcl error event. Handle the Tcl error with the `catch` command. This does not affect the XPM error condition.

xpmfetch

```
xpmfetch ?-warn var? xpmId field
```

This command retrieves data from the specified field within the XPM handle. Depending on the underlying record format, a single field string can imply multiple values. For example, an FRL field that is defined with multiple subfields.

- This command returns a list of datum handles, one per value: subfields for FRL, subcomponents for HL7.
- If any warning messages are generated during the retrieval, then they are appended to *var*, if specified.
- An error is returned if the field specification is invalid, with respect to the record definition. An error is also returned when any item's value cannot be retrieved. For example, the value is not present in XPM handle data-state, or the underlying message is too short to cover the field.

`field` is interpreted as an input value to an XPM operation. Its leading character identifies which side of the translation the data is retrieved from:

- `@name`: Temporary field name
- `~name`: Output field name
- `name`: Input field name

xpmmetaget

```
xpmmetaget ?-rw | -ro | -all? xpmId ?key?
```

Messages that are produced by `xpmId` inherit metadata from a common source. This command queries that default metadata.

- If a `key` value is not specified, then the available metadata keys are returned.
- If a `key` value is specified, then the metadata that is associated with that key returns.
- `-rw` and `-ro` restrict the list to the read-write (modifiable) and read-only key lists.

xpmmetaset

```
xpmmetaset xpmId key VALUE ?key2 VALUE2 ...?
```

This sets values for one or more metadata keys.

This command modifies the default metadata that is inherited by messages produced by `xpmId`.

This command returns an empty string.

xpmstore

```
xpmstore ?-list? xpmId field type args
```

This command stores values into the output fields of `xpmId`.

- If `-list` is used, then `args` is a list containing all of the values to store. Otherwise, the value list is all of the remaining arguments to the command.
- `field` is interpreted as an output field. A leading `@` identifies a temporary field variable. Input fields cannot be modified.
- `type` identifies the default item type for the list, although the datum can be without identification. Type choices are `c`, `d`, or `v`.

Store both strings and datum objects using this command.

- Constant strings use `-c`.
- Datum objects use `-d`.

- Special values use `-v`.

The only special value currently supported is null, which represents a value that is present, but empty. In HL7, for example, it is encoded as ""(two double-quotes). It is not treated specially in FRLs.

Constant string values default to type `ch` when stored. Specify an `hci` data type with a `-type datatype` trailer.

For example:

```
xpmstore xpm0 PatientName d datum0 datum1
xpmstore -list xpm0 {PatientName.[0,1]} d "datum0\ datum1"
xpmstore xpm1 0(0).PID(0).00041 d -c Smith datum2\
-v null
```

This command returns an empty string.

Generate routes

The system use generate routes to determine message routing at run-time. In Network Configurator, configure generate routes with one or more TPS procedures. Messages entering these procedures have a representation of the remaining routes attached to them. The procedures can query and modify these routes. `CONTINUE` disposition messages returned by the generate procedures are translated and routed according to the route configurations that are attached to them.

A system route-detail is described by a keyed list. The keys that are used in the keyed list depend on the route-detail type. For example, `generate`, `raw`, or `xlate`. The route-detail type determines the remaining keys:

- **Generate:** `PROCS`
- **Raw:** `DEST`, `PROCS`
- **Xlate:** `DEST`, `XLATE`, `PREPROCS`, `POSTPROCS`

`PROCS`, `PREPROCS`, and `POSTPROCS` are all TPS configurations, each having two sub-keys:

- **PROCS:** This is the ordered list of procedures making up the TPS.
- **ARGS:** This is the list of user-supplied arguments for each entry on the `PROCS` list.

These keys involve nested keyed lists.

- The value of `DEST` is a list of destination threads where the translated message is sent. Specify any number of thread names, although each element of the list must be shorter than 64 characters.
- The `XLATE` key gives the name of the XLT file which controls the translation.

Examples

- An example of generate routes is this list that describes a pass-through raw route:

```
{ TYPE raw } { PROCS {{ PROCS {} } { ARGS {} }}}}
```

- This raw route uses two TPS procedures; the first takes one user-supplied argument and the second takes two:

```
{DEST somedest}
{ TYPE raw } {PROCS {f
  { ARGS { arg1 { arg2-1 arg2-2 } } }
  { PROCS { proc1 proc2 } }
} }
```

- A route configuration is a list of these route detail lists.
This route configuration has two details: raw and xlate.

```
{
  {
    { PROCS {
      { ARGS { } }
      { PROCS { } }
    } }
    { TYPE raw }
  }
  {
    { PREPROCS {
      { ARGS postproc-arg }
      { PROCS postproc }
    } }
    { DEST somedest }
    { POSTPROCS {
      { ARGS preproc-arg }
      { PROCS preproc }
    } }
    { TYPE xlate }
    { XLATE some.xlt }
  }
}
```

Accessing and modifying the route configuration

Access and modify a message's route configuration with the `msgrouteget` and `msgrouteset` commands.

- `msgrouteget` returns the message's route configuration.
- `msgrouteset` replaces a message's entire route configuration with a new value. It returns an empty string.

For example:

```
Input: keylset newlist TYPE raw PROCS.ARGS {} \ PROCS.PROCS {}
Input: msgrouteset message3 {list $newlist}
Output:
Input: msgrouteget message3 fi {{TYPE raw} {PROCS {{ARGS {} } {PROCS {}}}}}
```

Note: These commands only work in the context of generate route TPS procedures. They error out if you invoke them in any other context. The examples in this text show how the commands work, even though you cannot run them interactively.

Protocol thread status

```
pdsetstatus status ?error_message?
```

status is one of:

- up
- opening
- error

error_message is an optional message for the thread protocol status.

This command is available only for UPoC protocols.

Tcl extension modifications

The `datcreate`, `grmfetch`, and `grmstore` Tcl extensions can handle XML node content.

datcreate

Note: `xml` is XML content in textual form, converted to DOM nodes when stored and retrieved from DOM nodes.

To create an XML element DOM node:

```
Input: datcreate "<xml attr1='value'> textvalue</xml>" xml
Output: datum0
Input: datget datum0 TYPE
Output: xml
Input: datget datum0 VALUE
Output: <xml attr1='value'> textvalue</xml>
```

To create an XML attribute DOM node:

```
Input: datcreate attr1='value' xml
Output: datum0

Input: datget datum0 TYPE
Output: xml

Input: datget datum0 VALUE
Output: attr1='value'
```

grmstore

XML is stored using the `x` datatype.

To store an XML string into an any node, use the `x` type:

```
grmstore $grmIdXml root.child.##any x "<xml attr1='value1'> childvalue</xml>"
```

Note: Currently, the `x` datatype can only be used on `##any` or `##anyAttributeNodes` and that `##any` or `##anyAttributeNodes` only accepts data with the `x` datatype.

When you must store multiple wildcard nodes to a wildcard field, you can list the multiple values as you would for `grmstore` in other formats. Only those addresses terminating in `##any` or `##anyAttribute` accept XML type nodes.

For example, two values were retrieved using:

```
Input: grmfetch $grmIdXml ns1:root.ns1:xml.##anyAttribute
Output: datum0 datum1
```

To store these two values there are these choices:

- Using the `x` type:

```
set dh1 [datget datum0 VALUE]
set dh2 [datget datum1 VALUE]grmstore $grmIdXml_2 root.xml.##anyAttribute x $dh1 $dh2
```

- Using the `datum` type:

```
grmstore $grmIdXml_2 root.xml.##anyAttribute d datum0 datum1
```

- Using the `c` type:

```
grmstore $grmIdXml_2 root.xml.##anyAttribute c attr1='123' -xml attr2='456' -xml
```

Engine NetConfig interface extensions

The engine NetConfig Interface (NCI) provides access to information that is stored in NetConfig files. Through this extension, you can load configurations that are saved in the NetConfig file from Tcl. This is accomplished in the same way that the engine loads these configurations during runtime.

Note: These commands and the data they return are primarily intended for internal use. The interfaces or data can change at any time.

The only command of NCI extensions is the command `netconfig`. Different queries can be accomplished by passing different actions and arguments to this command. The output also differs according to the requested action.

```
Input: netconfig get process list
Output: proc0 proc1 proc2
```

```
Input: netconfig get version
Output: 3.10
```

Loading a specified NetConfig

Usually, the NCI module does not require any initialization. Commands attempt to load the NetConfig file by themselves. To load a NetConfig file outside of your current site, you must first invoke the `netconfig load file` command.

```
Input: netconfig load ../helloworld/NetConfig
Output: 1
```

Querying information about threads

The items an NCI module provides are extracted from the NetConfig file contents or are computed based on that. You can get the total amount of threads set up in your current NetConfig file.

```
Input: netconfig get connection count
Output: 2
```

You can also list out all the names of those threads.

```
Input: netconfig get connection list
Output: sane insane
```

At this point, you know the exact names of the threads. You can get additional details of the configuration by calling:

```
netconfig get connection data <thread>
```

This gives you the entire keyed list of the configuration for that thread. Then, you can pick out what is necessary.

```
Input: set conndata [ netconfig get connection data sane]
Output:
  { AUTOSTART 1 }
  { BITMAP /hci/bitmaps/hcilogo.xbm }
  { COORDS {267 75} }
  { DATAFORMAT {
    { FRLTYPE offlen }
    { OFFLEN {
      { LEN 0 }
      { OFF 0 }
    } }
  } }
.....
  { STARTPROCS {
    { ARGS {} }
    { PROCS {} }
  } }
  { STOPPROCS {
    { ARGS {} }
    { PROCS {} }
  } }
  { SYMNAME sane }
  { USERCOVERDB 0 }
```

```
Input: keylget conndata PROTOCOL.OUTFILE
Output: output
```

Querying information about processes, groups, or destination threads

Querying information about processes, groups, and destination threads can be performed in the same manner as threads. The difference is that the NCI module does not provide a command to directly get the destination thread amount.

For processes you have one more option to get the threads included in that process. For groups, you get all options on processes, except that you cannot get data on a group. A group is a conceptual item in NetConfig and not a real list.

This is the provided command list:

```
get group connections <group>
get group count
get group list
get process connections <process>
get process count
get process data <process>
get process list
get destination data <dest>
get destination list
```

Except for retrieval information from NetConfig, the NCI module also provides judgment utilities for groups and processes. You can know whether a thread is in a group or process by directly calling the command:

```
ingroup <group> <thread>
inprocess <process> <thread>
```

Retrieving the NetConfig file version

When a system site is migrated from an earlier engine version, the NetConfig version might be different from the current engine's default version. Thus, some structures in it could be different.

The NCI module provides a command to help resolve this issue:

```
Input: netconfig get version
Output: 3.10
```

Detecting a NetConfig modification

NetConfig can be modified after a process starts. The update does not take effect until the process is restarted.

The NCI module provide a method to assist you during runtime to decide whether the NetConfig has been modified. If so, then restart the process.

```
Input: netconfig modified
Output: 0
```

NCI reference

All of the NCI functions are implemented under the `netconfig` command. This performs the requested job, based on the different actions and arguments that are passed to it.

This table shows a list of `netconfig` action args:

Action arguments	Description
<code>get connection count</code>	This returns the thread amount in the current Net-Config.
<code>get connection data thread</code>	This returns the thread configuration's keyed list of <i>thread</i> .
<code>get connection list</code>	This returns the thread name list in the current Net-Config.
<code>get group connections group</code>	This returns the thread name list in group <i>group</i> .
<code>get group count</code>	This returns the group amount in the current NetConfig.
<code>get group list</code>	This returns the group name list in the current Net-Config.
<code>get process connections process</code>	This returns the thread name list in process <i>process</i> .
<code>get process count</code>	This returns the process amount in the current Net-Config.
<code>get process data process</code>	This returns the process configuration's keyed list of <i>process</i> .
<code>get process list</code>	This returns the process name list in the current NetConfig.
<code>get destination data dest</code>	This returns the destination thread's configuration keyed list of <i>dest</i> .
<code>get destination list</code>	This returns the destination thread's name list in the current NetConfig.
<code>get version</code>	This returns the current NetConfig version.
<code>ingroup group</code>	thread This returns 1 if <i>thread</i> is in <i>group</i> ; otherwise, 0.
<code>inprocess process</code>	thre This returns 1 if <i>thread</i> is in <i>process</i> ; otherwise, 0.
<code>ad</code>	
<code>load ?file?</code>	This returns 1 if <code>load ?file?</code> is successful; otherwise, 0.
<code>modified</code>	This returns 1 if the current NetConfig file has been modified; otherwise, 0.

HTTP server

The HTTP server interface provides a way for the system to interact with a web server. HTTP server does not provide a web server. It is a plug-in to an existing web server, so it can communicate with the system.

Note: As long as your web server supports Perl CGI scripts, the system can connect to that web server.

There are three aspects of the HTTP server:

- Aspect #1: HTTP server as a web interface to the system

With HTTP server support, users can configure web servers to be client interfaces to the system. Internet browsers and other web-enabled applications can send queries, retrieve data, and send data to and from a system instance. It does this by interfacing with a web server client.

On the operating system, server-side CGI scripts can open a dynamic connection to a dedicated system protocol thread. When the connection is open, the scripts can pass messages to and from the engine.

The CGI scripts can be interfaced with one of these methods:

- Directly, through direct use of the HTTP protocol by a web-enabled client application.
 - Indirectly by embedding the CGI script into HTML through the use of HTML forms. This is ideal for a user-interface to system messages from a web browser.
- Aspect #2: HTTP server as an operating system web content provider
- The second is that HTTP server support enables the system to become a web content provider. The system already provides web content by transferring static HTML files using FTP or Fileset protocol. The HTTP server enables the system to interface with a web server. This becomes another tool for the web developer to provide dynamic content.
- When calling a system server plug-in from within a web form, web developers can access data from various sources. For example, databases or existing legacy applications. This is useful when paired with the system XML parser. When the XML translation is used, data of virtually any source or format can become web content in XML format.
- Aspect #3: HTTP server as a gateway through the firewall
- The third aspect is security. To pass data in and from firewalls, a web server may be the only avenue for data traffic. By configuring a system server plug-in, remote users can direct the data to the system by posting to the system server script. This provides safe passage through the firewall. The same is performed for data retrieval, where the system returns queried data from the firewall.

HTTP server interface

The HTTP server takes the form of server-side CGI scripts, including `CL_Tcp` and `CL_File`.

CL_Tcp

This multi-purpose script is capable of both sending and retrieving data to and from the system. When called, the script performs these actions:

- Accepts input data from the client.

- Opens a TCP/IP connection to a pre-configured system protocol thread.
- Sends the input data to the system.
- Receives a response/reply.
- Returns the response/reply back to the client.

This script is considered multi-purpose because it is used for these basic functions:

- Sending data to the system and receiving a reply.
- Sending a request to the system and receiving back the response.

The script is also designed to permit multiple clients to run it simultaneously. With this feature, users can specify a range of socket ports. When a socket port is unavailable, due to another instance of the script already running, the next socket port is attempted. This happens until a port is found. You can also specify a time-out value, that, if exceeded, causes the script to give up and return a time-out error to the client.

For example, the `CL_TcpQuery` script is configured with a socket range of 9001 to 9005. The system site is also configured with five TCP/IP protocol threads, all of which are servers and are listening on ports 9001 to 9005. With this configuration the web server can support a maximum of five simultaneous web client connections to the system through the `TcpQuery` script. If a sixth connection is attempted, then the client would receive a time-out error if no connection is released before the time-out interval expires.

Arguments are:

- `timeout`
Optional. This specifies how much time, in seconds, the script can spend attempting to connect to system threads before returning a time-out error.
- `data`
Required. This specifies the data to pass to the system.

CL_File

This script is used only for sending and receiving static documents to a web server. It does this by transferring documents from a web client to an intermediary web server. Then, a system instance can retrieve or deliver the files using the Fileset protocol driver.

Script inputs

Script inputs include:

- `method`
Required for all. Specifies the action to take:
 - `get` retrieves a file from the specified path.
 - `put` sends a file to the specified path.
 - `dir` requests a list of files in the specified directory path.
- `mode`
Optional.

- For the `put` method, this indicates whether data is written to a destination file in ASCII or binary mode.
- For the `get` method, this indicates how data is read from a source file. The default is binary.
- `path`
Required. Specifies the file path relative to the root web directory on the server.
 - For the `put` method, this is the destination path to the file that is created on the server.
 - For the `get` method, this is the source path to the file that is read from the server.
 - For the `dir` method, this is the directory path that is read.
- `style`
Optional. Used only with the `dir` method. Specifies the format of the outputted directory listing. The default is `short`.
 - `short` returns file names only, similar to the UNIX `ls` command. Default value.
 - `long` returns file names, dates, owners, and permissions, similar to the UNIX `ls -l` command.
- `data`
Required. This is applicable only for the `put` method. Specifies the data to pass to the system.

Server-side configuration

Use an `ini` file to define the configurable parameters. The scripts use the `CL_HTTPServer.ini` file.

Potentially configurable parameters within `CL_HTTPServer.ini` include:

- `CL_Tcp`. This contains:
 - `host`: The system location.
 - `Socket_Set` is the range of sockets to use for connection to the system. For example `9001-9005, 9002, 9004, 9006, or 9001, 9003-9005`.
 - `timeout` is the default time, in seconds, to spend attempting to connect before returning a time-out error. This is overridden by the `script` argument, if one is specified.
 - `PacketStyle` can be `m1p` or `len10`. For example, `packetStyle=len10`.
`m1p` is the HL7 MLP protocol.
`len10` has ASCII length = 10 and Exclusive Fill = 0.
- `CL_File`. This contains:
 - `Dir`: The full path to the directory on the system machine to which files are transferred.
 - `Del`: A `y` or `n` value that indicates whether files are deleted from the server after they are retrieved by a client. This applies only to the `get` method.

HTTP server usage

The HTTP server scripts are designed for use by different types of web clients. It accomplishes this by accepting inputs in different forms, thereby better accommodating different clients.

Web browsers and forms: multi-part/form-data format

If the client is a web browser, then HTTP server scripts can interact with HTML forms and accept input from them in multi-part/form-data encoding format.

Example:

```
<html><head>
<title>test page for testing CL_Tcp server script
</title>
</head><body>
<p>CL_Tcp.pl</p>
<form action="/cgi-bin/CL_Tcp.pl" enctype="multipart/form-data"
method="POST">
<p>timeout: <input type="text" name="timeout"></p>
<p>data file: <input type="file" name="data"></p>
<input type="submit" value="Submit">
</form>
</body></html>
```

This example uses two input fields, **timeout** and **data**, for providing input arguments to the CL_Tcp script. The form uses the file input type, where you can select **Browse** to interface with your local file system.

The form also uses the multi-part/form-data encoding type, where the server script can accept multiple data inputs of almost any type. For example, although the data input may be binary, it can be safely passed along with the time-out input, which is text.

Web-enabled applications: tagged format

The web-enabled application is another type of HTTP client that may require to communicate with the HTTP server. This application can open its own HTTP connection to a remote web server and send data directly to it. It does this without going through web pages, forms, or browsers.

This type of application may find it less convenient to use multi-part/form-data encoding to format the data and input parameters. Therefore, an alternate, more application-friendly format is provided: the tagged format.

In the tagged format, the same inputs are used, but they are presented to the HTTP server scripts in a different way. This format uses tagged values that are similar to XML/HTML to indicate all input options, except for the data itself. The data is passed immediately after the tagged inputs.

This defines the rules for the tagged input format:

```
<CL_CONX>
  <Option 1=Value1>
  <Option 2=Value2>
  .
  .
  <Option n=Valuen>
</CL_CONX>
```

[Data]

The option and value pairs are the same as those defined in another topic. See [Script inputs](#). All whitespace between tags are ignored, so the newlines and tabs in the above example are for readability purposes only and not required.

The entire `CL_CONX` construct can be completely omitted if all the inputs are optional and only the data portion is required. This is only the case for `CL_Tcp`. In this case, the application could pass the data as-is in the Body portion of the HTTP command.

These are the usage formats for the various HTTP server scripts:

- All non-italic characters are literal.
- All italic characters are non-literal.
- Optional values are enclosed in brackets ([]).
- Brackets with a pipe symbol ([|]) denote a choice.

CL_File

`CL_File` methods include:

- `get` method

```
<CL_CONX>
  <Method=get>
  <Path=RemoteSourceFilePath>
  [<Mode= [binary | ascii]>]
</CL_CONX>
```

- `put` method

```
<CL_CONX>
  <Method=put>
  <Path=RemoteDestFilePath>
  [<Mode= [binary | ascii]>]
</CL_CONX>
[Data]
```

- `dir` method

```
<CL_CONX>
  <Method=dir>
  <Path=RemoteDirPath>
  [<Mode= [binary | ascii]>]
  [<Style= [short | long]>]
</CL_CONX>
```

- `CL_Tcp`

```
<CL_CONX>
  [<Timeout= secs>]
</CL_CONX>
]
[Data]
```

grmfetch

The system can only retrieve values in terminal data nodes:

```
Input: grmfetch $grmIdXml root.child.&attr1
Output: datum0
Input: datget datum0 TYPE
Output: ch
Input: datget datum0 VALUE
Output: definedAttrValue
```

Data is returned as an XML Datum object when `grm` is created for an XML document and data is retrieved from an address ending in `##any` or `##anyAttribute`. This object is the textual representation of the matching wildcard nodes and their children.

For example:

```
Input: grmfetch $grmIdXml root.child.0(0).##any
Output: datum1

Input: datget datum1 TYPE
Output: xml

Input: datget datum1 VALUE
Output: <xml attr1='value'>textvalue</xml>
```

If the selected path is explicitly to `##anyAttribute`, multiple values are returned if multiple values match the wildcard.

For example:

```
Input: grmfetch $grmIdXml root.##anyAttribute
Output: datum0 datum1

Input: datget datum0 TYPE
Output: xml

Input: datget datum0 VALUE
Output: attr1="attrvalue1"

Input: datget datum1 TYPE
Output: xml

Input: datget datum1 VALUE
Output: attr2="attrvalue2"
```

X12 procedures

X12 procedures include:

- `joinX12`
The `joinX12` procedure is most commonly found in an outbound TPS, and is preceded by `x12MetaData`, if required.
- `splitX12`

Translations are defined between specific transaction types. For example, a purchase order or functional acknowledgment. Because an interchange can contain multiple functional groups with different types, split the message before translation. The system can handle group-level splits or transaction set-level splits. Tcl procedures are supplied to perform these splits.

joinX12

To send out a transaction set or group, the system must build up a valid X12 interchange using the `joinX12` procedure.

The `joinX12` procedure:

- Examines the start of the message to determine whether it is groups or transaction sets.
- Takes a message, a group or a transaction set, and uses the user-defined metadata to add the appropriate ISA, IEA, GS, and GE segments.

The `joinX12` procedure requires:

- The input message to have the correct user-defined metadata to populate the ISA, and optional GS, segments.
- The correct `SEPCHARS` metadata to be defined.

There are no arguments, so if any metadata is missing or incorrectly formatted, an error results and the message goes to the error database. This procedure does not generate a new message. It modifies the original, if successful, so that all attributes of the original, such as the message in the recovery database, are maintained.

Metadata setup

A `X12MetaData` template is provided in the Script Editor for setting up the metadata. This procedure is placed before `joinX12` to fill in the metadata fields. The fields are commented to indicate their purpose.

X12MetaData

This is a template that is used to create TPS procedures to put the correct user-defined metadata on a message.

Note: If the `splitX12` procedure was used, then the metadata fields have already been populated. The `X12MetaData` template can be used to change values before building the outbound message, if necessary.

`X12MetaData` is used in the outbound TPS before `joinX12`, and:

- Tests non-X12 to X12 translations.
- Formats fields set by the user.
- Keeps the current value from user-defined metadata if the field is not formatted by the user.
- Uses a default value if the field is not defined by the user or by user-defined metadata.

Fields are commented to indicate their purpose.

When using this template, remember these points:

- To set one of the `GS` or `ISA` fields, uncomment the corresponding set command and fill in your value.
- If you do not set a `GS` or `ISA` field and your message has a value assigned in user-defined metadata, then the message value is used.
- If you do not set a value and your message has no user-defined metadata, then the default value is used.
- If you are translating from version 3020 to 4010, then the user-defined metadata is filled in by `splitX12` on the inbound side.
- When you build the interchange, set the `ISA` version number to 00401 and the `GS` version to 004010.

All other values are the same as the inbound message.

If your message is split at the group level, then the group user-defined metadata is not used by `joinX12`. The `GS` and `GE` are translated as normal segments in the message. They are part of the message data. Therefore, any `GS` values that are set in `X12MetaData` procs do not have any effect.

splitX12

The main procedure is a TPS called `splitX12`. By default, it splits an interchange into functional groups.

The `splitX12` procedure is most commonly found in an inbound TPS, and:

- Attaches the `ISA` and `GS` data as user-defined metadata, as a keyed list.
- Defines the `SEPCHARS` metadata.
- Generates messages that use the recovery database. This is when **Use Recovery Database** is enabled on the original message.

The `SEPCHARS` metadata field key in the message object overrides default separator characters on a per-message basis.

If the metadata field is `NULL`, then the defaults are used to parse or encode the message. If set, then it is a keyed list.

These keys are searched for in the `Userdata` metadata field:

- `FIELD`
- `COMPONENT`
- `REPEAT`
- `ESCAPE`
- `SEGMENT`
- `DECIMAL`

Transaction split

Using the argument `ST`, `splitX12` performs a transaction set split. Most groups contain many transaction sets, so an `ST` split results in additional, smaller messages.

A group split results in fewer, larger messages.

Group split

A group split requires a different message definition than a transaction split. The group-defined message must define the transaction set as a repeating group surrounded by GS and GE. Standard X12 messages are transaction-defined. To use a group split, you must create a variant that surrounds the transaction set with GS and GE. Translation of a group-defined message always iterates over the transaction sets.

Required fields are encoded in user-defined metadata. This metadata is stored in the split messages. This user-defined metadata is in the form of a keyed list. It contains at least an ISA key with the list of ISA fields as the value.

For example:

```
ISA*00*      *00*      *ZZ*CONAME  *ZZ*HDX      *000515*2347*U*00304*000000001*0*P*^~
```

This interchange would have minimum user-defined metadata and any other fields in the keyed list.

```
{{ISA {00 "" 00 "" ZZ CONAME ZZ HDX 000515 2347 U 00304 000000001 0 P ^}}}
```

Transaction ID determination uses this ISA metadata so it should be included. Store separator characters in the message metadata.

Note: A message must contain only one interchange. If it contains more interchanges, then an error results.

Group-defined messages use this structure:

```
GS
{
    ST
    . . .
    SE
}
GE
```

Transaction set messages use this form:

```
ST
. . .
SE
```

hciX12splitinterchange Tcl proc

This procedure is included with system installs and is found in `\integrator\tclprocs\edi.tcl`. This procedure sends the argument to `x12:splitX12`. GS is group messages and ST is transaction set messages.

If `x12:splitX12` does not get any arguments, then the default value is GS.

```
Name:          hciX12splitinterchange
# Purpose:      Splits up an X12 interchange into groups or transaction sets.
# UPoC type:    tps
# Args:         tps keyedlist containing these keys:
#               MODE    run mode ("start", "run" or "time")
#               MSGID   message handle
#               ARGS    user-supplied arguments:
```

```
# GS: split into groups (default)
# ST: split into transaction sets
```

To use this procedure, you must correctly set the `GS/ST` arguments. This is configured from the `xlt` file or the GUI.

In the `xlt` file, set `GS/ST` in this way:

```
{ ARGS GS }
{ PROCS hciX12splitinterchange }
{ THREADED 0 }
```

In the GUI, `GS/ST` can be entered in the TPS Properties dialog box's **Args** text box.

Saving headers with user-defined metadata

After the messages are split, they can be translated. The segments that were stripped off, the ISA and possibly `GS`, could be required later. They are stored in the user-defined metadata of each split message. These segments are stored as a keyed list whose value is the list of elements. The user-defined metadata of each message that is generated by `splitX12` contains an `ISA` and possibly a `GS`. This depends on whether it was a group- or transaction-level split.

This makes the `ISA` and `GS` header fields available to procedures later on. To retrieve them:

```
set user_data [msgmetaget $mh USERDATA]
set isa_data [keylget user_data ISA]
set isa01 [lindex $isa_data 1]
```

Group-defined transaction sets

In the X12 Configurator, you can define transaction sets as `GS...GE`, in addition to `ST...SE`. This type of transaction set is required for a group-level split. The group variant must start with `GS` and end with `GE`, and should include the transaction set as a repeating group.

The group variant uses this form:

```
GS
{
    ST
    . . .
    SE
}
GE
```

Note: Do not use curly braces around everything to set the transaction message to repeat. This type of message crashes the engine.

X12 TrxID determination

The X12 trxID now includes the receiver's and sender's ID and ID qualifiers, and requires the user-defined metadata as generated by `splitX12`.

The new trxID uses the form:

```
MMM_RECEIVER_RQ_SENDER_SQ
```

- `MMM` = message number (that is, 835 from the transaction set)
- `RECEIVER` = receiver ID from the ISA
- `RQ` = receiver ID qualifier from the ISA
- `SENDER` = sender ID from the ISA
- `SQ` = sender ID qualifier from the ISA

The trxID determination works for transaction set messages (`ST`) or group messages (`GS`). The trxID determination relies on the correct separator characters in the `SEPCHARS` metadata field. It also relies on the ISA user-defined metadata.

If the ISA user-defined metadata is missing from the message, then the `RECEIVER`, `RQ`, `SENDER`, and `SQ` fields are blank and the underscores are omitted. The trxID would be 835. Routing is similar to HL7, where unspecified trailing fields are ignored.

For example, in the trxID `835_MSGGROUP_ZZ_COLVERLEAF_ZZ`, 835 would route all 835 transactions, and `835_MSGGROUP` would route all 835s bound for MSGGROUP. This handles most routing scenarios without the requirement for a custom trxID procedure.

Note: If X12 trxID determination is specified for the inbound thread, then the `splitX12` is automatically included in the inbound TPS with default arguments.

HL7 message types (functional areas)

These codes are based on the HL7 v2.3 standard.

For more information about HL7, refer to *Health Level Seven - An Application Protocol For Electronic Data Exchange In Healthcare Environments*. This is available from Health Level Seven, Inc.

This table lists the HL7 message types:

Message type	Description	Location
ACK	General acknowledgment	Chapter 2: Control/Query
ADR	ADT (Admission/Discharge/Transfer) response	Chapter 3: Patient Administration
ADT	ADT message	Chapter 3: Patient Administration
ARD	Ancillary report	Chapter 7: Observation Reporting

Message type	Description	Location
BAR	Add/change billing account	Chapter 6: Financial Management
CRM	Clinical study registration	Chapter 7: Observation Reporting
CSU	Unsolicited clinical study review	Chapter 7: Observation Reporting
DFT	Detail financial transaction	Chapter 6: Financial Management
DSR	Display response	Chapter 2: Control/Query
EDR	Enhanced delay response	Chapter 2: Control/Query
ERP	Event replay response	Chapter 2: Control/Query
EQQ	Embedded query language query	Chapter 2: Control/Query
MCF	Delayed acknowledgment	Chapter 2: Control/Query
MDM	Documentation message	Chapter 9: Medical Records/Information Management
MFN	Master files notification	Chapter 8: Master Files
MFK	Master files application acknowledgment	Chapter 8: Master Files
MFD	Master files delayed application acknowledgment	Chapter 8: Master Files
MFQ	Master files query	Chapter 8: Master Files
MFR	Master files query response	Chapter 8: Master Files
ORF	Observation result/record response	Chapter 7: Observation Reporting
ORM	Order message	Chapter 4: Order Entry
ORR	Order acknowledgment message	Chapter 4: Order Entry
ORU	Observation result/unsolicited	Chapter 7: Observation Reporting
OSQ	Order status query	Chapter 4: Order Entry
OSR	Order status response	Chapter 4: Order Entry
PEX	Product experience	Chapter 7: Observation Reporting
PGL	Patient goal	Chapter 12: Patient Care
PIN	Patient insurance information	Chapter 11: Patient Referral
PPG	Patient pathway (goal-oriented) message	Chapter 12: Patient Care
PPP	Patient pathway (problem-oriented) message	Chapter 12: Patient Care

Message type	Description	Location
PPR	Patient problem	Chapter 12: Patient Care
PPT	Patient pathway (goal-oriented) response	Chapter 12: Patient Care
PPV	Patient goal response	Chapter 12: Patient Care
PRR	Patient problem response	Chapter 12: Patient Care
PTR	Patient pathway (problem-oriented) response	Chapter 12: Patient Care
QCK	Query general acknowledgment	Chapter 2: Control/Query
QRY	Query, original mode	Chapter 2: Control/Query
RAR	Pharmacy administration information	Chapter 4: Order Entry
RAS	Pharmacy administration message	Chapter 4: Order Entry
RCI	Return clinical information	Chapter 11: Patient Referral
RCL	Return clinical list	Chapter 11: Patient Referral
RDE	Pharmacy-encoded order message	Chapter 4: Order Entry
RDR	Pharmacy dispense information	Chapter 4: Order Entry
RDS	Pharmacy dispense message	Chapter 4: Order Entry
REF	Patient referral	Chapter 11: Patient Referral
RER	Pharmacy-encoded order information	Chapter 4: Order Entry
RGV	Pharmacy give message	Chapter 4: Order Entry
RGR	Pharmacy dose information	Chapter 4: Order Entry
ROR	Pharmacy prescription order response	Chapter 4: Order Entry
RPA	Return patient authorization	Chapter 11: Patient Referral
RPI	Return patient information	Chapter 11: Patient Referral
RPL	Return patient display list	Chapter 11: Patient Referral
RPR	Return patient list	Chapter 11: Patient Referral
RQA	Request patient authorization	Chapter 11: Patient Referral
RQC	Request clinical information	Chapter 11: Patient Referral

Message type	Description	Location
RQI	Request patient information	Chapter 11: Patient Referral
RQP	Request patient demographics	Chapter 11: Patient Referral
RRA	Pharmacy administration acknowledgment	Chapter 4: Order Entry
RRD	Pharmacy dispense acknowledgment	Chapter 4: Order Entry
RRE	Pharmacy encoded order acknowledgment	Chapter 4: Order Entry
RRG	Pharmacy give acknowledgment	Chapter 4: Order Entry
RRI	Return patient referral	Chapter 11: Patient Referral
SIU	Schedule information unsolicited	Chapter 10: Scheduling
SPQ	Stored procedure request	Chapter 2: Control/Query
SQM	Schedule query	Chapter 10: Scheduling
SQR	Schedule query response	Chapter 10: Scheduling
SRM	Schedule request	Chapter 10: Scheduling
SRR	Schedule request response	Chapter 10: Scheduling
SUR	Summary product experience report	Chapter 7: Observation Reporting
TBR	Tabular data response	Chapter 2: Control/Query
UDM	Unsolicited display message	Chapter 2: Control/Query
VQQ	Virtual table query	Chapter 2: Control/Query
VXQ	Query for vaccination record	Chapter 4: Order Entry
VXX	Vaccination query response with multiple PID matches	Chapter 4: Order Entry
VXR	Vaccination query record responses	Chapter 4: Order Entry
VSU	Unsolicited vaccination record update	Chapter 4: Order Entry

HL7 segment ID

These codes are based on the HL7 v2.3 standard.

For more information about HL7, refer to *Health Level Seven - An Application Protocol For Electronic Data Exchange In Healthcare Environments*. This is available from Health Level Seven, Inc.

This table lists the HL7 segment IDs:

Segment ID	Description	Location
ACC	Accident	Chapter 6: Financial Management
ADD	Addendum	Chapter 2: Control/Query
AIG	Appointment information, general resource	Chapter 10: Scheduling
AIL	Appointment information, general resource	Chapter 10: Scheduling
AIP	Appointment information, person-nel resource	Chapter 10: Scheduling
AIS	Appointment information, service	Chapter 10: Scheduling
AL1	Patient allergy information	Chapter 3: Patient Administration
APR	Appointment preferences	Chapter 10: Scheduling
ARQ	Appointment request	Chapter 10: Scheduling
AUT	Authorization information	Chapter 11: Patient Referral
BHS	Batch header	Chapter 2: Control/Query
BLG	Billing	Chapter 4: Order Entry
BTS	Batch trailer	Chapter 2: Control/Query
CDM	Charge description master	Chapter 8: Master Files
CM0	Clinical study master	Chapter 7: Observation Reporting
CM1	Clinical study phase master	Chapter 7: Observation Reporting
CM2	Clinical study schedule master	Chapter 7: Observation Reporting
CSP	Clinical study phase	Chapter 7: Observation Reporting
CSR	Clinical study registration	Chapter 7: Observation Reporting
CSS	Clinical study data schedule	Chapter 7: Observation Reporting
CTD	Contact data	Chapter 11: Patient Referral
CTI	Clinical trial identification	Chapter 7: Observation Reporting
DB1	Disability	Chapter 3: Patient Administration
DG1	Diagnosis	Chapter 6: Financial Management
DRG	Diagnosis related group	Chapter 6: Financial Management

Segment ID	Description	Location
DSC	Continuation pointer	Chapter 2: Control/Query
DSP	Display data	Chapter 2: Control/Query
EQL	Embedded query language	Chapter 2: Control/Query
EQR	Event replay query	Chapter 2: Control/Query
ERR	Error	Chapter 2: Control/Query
EVN	Event type	Chapter 3: Patient Administration
FAC	Facility	Chapter 7: Observation Reporting
FHS	File header	Chapter 2: Control/Query
FT1	Financial transaction	Chapter 6: Financial Management
FTS	File trailer	Chapter 2: Control/Query
GOL	Goal detail	Chapter 12: Patient Care
GT1	Guarantor	Chapter 6: Financial Management
IN1	Insurance	Chapter 6: Financial Management
IN2	Insurance additional information	Chapter 6: Financial Management
IN3	Insurance additional information, certification segment	Chapter 6: Financial Management
LCC	Location charge code	Chapter 8: Master Files
LCH	Location characteristic	Chapter 8: Master Files
LDP	Location department	Chapter 8: Master Files
LOC	Location identifier	Chapter 8: Master Files
LRL	Location relationship	Chapter 8: Master Files
MFA	Master file acknowledgment	Chapter 8: Master Files
MFE	Master file entry	Chapter 8: Master Files
MFI	Master file identification	Chapter 8: Master Files
MRG	Merge patient information	Chapter 3: Patient Administration
MSA	Message acknowledgment	Chapter 2: Control/Query
MSH	Message header	Chapter 2: Control/Query
NK1	Next of kin/associated parties	Chapter 3: Patient Administration

Segment ID	Description	Location
NPU	Bed status update	Chapter 3: Patient Administrations
NTE	Notes and comments	Chapter 2: Control/Query
OBR	Observation Entry	Chapter 4: Order Entry Chapter 7: Observation Reporting
OBX	Observation/result	Chapter 7: Observation Reporting Chapter 9: Medical Records/Information Management
ODS	Dietary orders, supplements, and preferences	Chapter 4: Order Entry
ODT	Diet tray instructions	Chapter 4: Order Entry
OM1	General segment (fields that apply to most observations)	Chapter 8: Master Files
OM2	Numeric observation	Chapter 8: Master Files
OM3	Categorical test/observation	Chapter 8: Master Files
OM4	Observations that require specimens	Chapter 8: Master Files
OM5	Observations batteries (sets)	Chapter 8: Master Files
OM6	Observations that are calculated from other observations	Chapter 8: Master Files
ORC	Common order	Chapter 4: Order Entry
ORO	Order other	Chapter 4: Order Entry
PCR	Possible causal relationship	Chapter 7: Observation Reporting
PDC	Product detail country	Chapter 7: Observation Reporting
PD1	Patient additional demographic	Chapter 3: Patient Administrations
PEO	Product experience observation	Chapter 7: Observation Reporting
PES	Product experience sender	Chapter 7: Observation Reporting
PID	Patient identification	Chapter 3: Patient Administrations
PRA	Practitioner detail	Chapter 8: Master Files
PRB	Problem detail	Chapter 12: Patient Care
PRC	Pricing	Chapter 8: Master Files

Segment ID	Description	Location
PRD	Provider data	Chapter 11: Patient Referral
PR1	Procedures	Chapter 6: Financial Management
PSH	Product summary header	Chapter 7: Observation Reporting
PTH	Pathway	Chapter 12: Patient Care
PV1	Patient visit	Chapter 3: Patient Administrations
PV2	Patient visit, additional information	Chapter 3: Patient Administrations
QAK	Query acknowledgment	Chapter 2: Control/Query
QRD	Query definition	Chapter 2: Control/Query
QRF	Query filter	Chapter 2: Control/Query
RDF	Table row definition	Chapter 2: Control/Query
RDT	Table row data	Chapter 2: Control/Query
RF1	Referral information	Chapter 11: Patient Referral
RGS	Resource group	Chapter 10: Scheduling
ROL	Role	Chapter 12: Patient Care
RQD	Requisition detail	Chapter 4: Order Entry
RQ1	Requisition detail 1	Chapter 4: Order Entry
RXA	Pharmacy/treatment administration	Chapter 4: Order Entry
RXC	Pharmacy/treatment component order	Chapter 4: Order Entry
RXD	Pharmacy/treatment dispense	Chapter 4: Order Entry
RXE	Pharmacy/treatment encoded order	Chapter 4: Order Entry
RXG	Pharmacy/treatment give	Chapter 4: Order Entry
RXO	Pharmacy/treatment order	Chapter 4: Order Entry
RXR	Pharmacy/treatment route	Chapter 4: Order Entry
RX1	Pharmacy order	Chapter 4: Order Entry
SCH	Schedule activity information	Chapter 10: Scheduling
SPR	Stored procedure request definition	Chapter 2: Control/Query

Segment ID	Description	Location
STF	Staff identification	Chapter 8: Master Files
TXA	Transcription document header	Chapter 9: Medical Records/Information Management
UB1	UB82 data	Chapter 6: Financial Management
UB2	UB92 data	Chapter 6: Financial Management
URD	Results/Update definition	Chapter 2: Control/Query
URS	Unsolicited selection	Chapter 2: Control/Query
VAR	Variance	Chapter 12: Patient Care
VTQ	Virtual table query request segment	Chapter 2: Control/Query

HL7 type codes

These codes are based on the HL7 2.1 version standard.

This table lists the HL7 type codes:

Type code	Description	Example
AD	Address: The street number or mailing address of a person or institution.	10 ASH LN^#3^LIMA^OH^48132
CE	Coded Element: Transmits codes and the text associated with the code.	54.21^Laproscopy^I9C^42112^^AS4
CK	Check Digit: Composite ID with check digit	128952^6^M11^ADT01
CM	Composite: A field that is a combination of other meaningful data fields. Each portion is called a component.	12372^RIGGINS^JOHN^""^MD
CN	Coded Name: Composite ID number and name.	12372^SMITH^JOHN^""^""^""^MD^ADT1
CQ	Coded Quantity: Composite quantity with units.	123.7^kg
DT	Date: Includes a 4-digit year and is formatted as ccyyymmdd.	19931104

Type code	Description	Example
FT	Formatted text: Text string embedded formatting instructions. These instructions are limited to those that are intrinsic and independent of the field's circumstances.	\.fi this is example text \.nf
ID	Identifier: Coded value in the form of an ST field.	20170
NM	Numeric: Can include a leading + or - sign. If no sign is present, then a positive number is assumed. Decimal points can be included. If no decimal point is present, then it is assumed to be an integer.	-123.456
PN	Person name: A person's name. This includes the given name, middle name, family name, prefix (for example, DR), suffix (for example, JR or III), and degree (for example, MD).	Smith^Larry^V
SI	Sequence ID: A positive integer	12345
ST	String: Any display characters can be used. String data is left-justified with optional trailing blanks.	This is a string
TM	Time: 24-hour clock notation, formatted as hhmm[ss[-zzzz]] in v2.1, and hhmm[ss[.ssss]][+/-zzzz] in v2.2 and v2.3. The default time for optional elements is local time of the sender. Seconds and time zone are optional.	132752-0600
TN	Telephone number	(214)-555-2000
TS	Time stamp: Contains the exact time of an event, including the date and time. The time zone may be sent as an offset from the coordinated universal time. If the time zone is not present, then it is the local time zone of the sender.	19931104132752-0600
TX	Text: String data meant for user display (on a terminal or printer).	Some highlighted text

Metadata fields

Message metadata is a set of information that the system engine must maintain along with each message as it passes through the system. Metadata is not part of the user data contained in a message. This section lists the various metadata fields.

This table lists the message metadata fields as they were originally implemented:

Metadata field	Description
Class	This reflects whether the message is currently an engine message or a protocol message.
DestThread	<p>This is the name of the current destination of the message.</p> <p>After the message is initially routed, the DestThread and the OrigDestThread are initialized to be the same value.</p>
DriverControl	<p>This contains a Tcl keyed list that gives you some control over the protocol driver.</p> <p>This field is a keyed list. It contains information that is specific to a single driver and other data that is generic and applies to almost all drivers.</p>
ExpectReply	<p>This is used during the protocol write stage to indicate if the engine should expect an application-level reply from this message.</p> <p>Before the first module in an outbound TPS stack gets the outbound engine message, this value is set to the destination connection's Await replies configuration.</p>
Forwarded	<p>This is a boolean value used to indicate if the message has ever been forwarded. This is useful if an outbound TPS stack module gets forwarded messages from another connection and must manipulate them as they arrive in the module.</p> <p>After turned on, this flag is never cleared by the engine.</p> <p>Messages created by copying a message inherit this value.</p>
Mid	<p>When a message enters the engine, it is assigned a unique message ID that is stored in the Mid field.</p> <p>This ID is used to identify and track the message as it flows through the engine, the recovery database, and possibly the error database.</p>

Metadata field	Description
OrigDestThread	This is the name of the original destination thread for the message after translation. This never changes throughout message life.
OrigSrcThread	This is the name of the thread or connection from which the message was originally read. This never changes throughout message life.
Priority	<p>This is used to give special treatment to a message. Each message is placed in an engine queue according to its priority.</p> <p>Higher priority messages are placed nearer to the head of each queue than messages of lower priority. Among messages of the same priority, a FIFO queue ordering is maintained.</p>
RecordFormat	<p>This is added by the translation system when creating an outbound message. This field contains the hub name, root, and name of format.</p> <p>For fixed and variable layouts, it gives the file name of the layout.</p> <p>For HL7 and other variant-based message formats, it gives the variant name.</p>
SkipXlate	<p>If this field is non-empty, then the translation thread does not route or translate the message. Instead, the translation thread sends a copy of the message to each destination thread that is listed in the source message's DestThread field.</p> <p>The translation thread does this by removing it from the pre-translation queue and placing it directly into the protocol queue one or more times. A copy is placed in the protocol queue for each destination. Each copy has the DestThread field updated to reflect the single destination to which the copy is sent.</p> <p>After all the copies have been created, the source message is removed from the recovery database. The destination messages are moved to the appropriate post-translation queues.</p>
SrcMid	<p>This is used to identify and track the pre-translation message that created a post-translation message.</p> <p>Every post-translation message that is created by the translation process has this field set to the Mid of the source message.</p> <p>In addition, copied messages inherit this value.</p>

Metadata field	Description
SrcThread	<p>This is the name of the current source of the message.</p> <p>This field and the DestThread field can change as a message is forwarded, to reflect the true current source and destination threads for the message.</p> <p>When a message enters the engine, its SrcThread and OrigSrcThread are initialized to be the same originating thread.</p>
Type	This reflects whether the message is a data or reply message.
XlateThread	This is the name of the translation thread that handled the message translation.

System-level environment variables

This table lists the system-level variables that are applied to every system instance:

Variable	Description
CL_INSTALL_DIR	The location where system products are installed.
HCIRoot	The home (root) directory of the system.
HCIVERSION	The version number of the system.
HCILICFILE	The location of the system license files.

msgXSLT

This transforms an XSLT on an XML message handle in Tcl. The input message is transformed using the given XSLT file. It then returns the output in a new message handle. The new handle inherits the metadata from the input message handle.

```
msgXSLT msgId xsltfn ?key VALUE? ?key2 VALUE2 ...?
```

- `msgId` is the message handle of the input data, and must contain a valid XML message.
- `xsltfn` is the file name of the XSLT file.
- `key VALUE` is the set of runtime parameters to pass to the XSLT engine.

Data Integrator

Data Integrator connects database management systems (DBMS) within the Cloverleaf healthcare integration environment. Data Integrator leverages the Open Database Connectivity (ODBC) standard. This lets Cloverleaf securely access and share data with industry-leading DBMSs.

The ODBC Tcl extensions are used to code the ODBC API within Tcl scripts wherever Tcl interpreters are available.

See [ODBC Tcl extensions](#).

The Tcl API is similar to the C API. One set of ODBC documentation can be used for both APIs, and existing ODBC C applications can be ported to Tcl. There are several generic differences between the C ODBC API and these Tcl extensions.

See [ODBC Tcl API and C API differences](#).

ODBC Tcl extensions

The ODBC Tcl extensions are used to code the ODBC API within Tcl scripts wherever Tcl interpreters are available. This ODBC functionality is most frequently used from within the engine and the HCITCL stand-alone Tcl interpreter.

The ODBC Tcl extension relies on middleware supplied by DataDirect Technologies. This middleware provides an ODBC development environment, the ODBC header files. It also contains software that is used during runtime when connecting the ODBC application to the DBMS (Database Management System).

DataDirect Technologies provides DataDirect Connect for this purpose.

The ODBC Tcl extensions have been converted to a dynamically-linked stubbed Tcl package.

"Dynamically-linked" means that it is not a part of any system binary. It is loaded into a running program when it is first required.

"Stubbed" means that calls to the Tcl API are made indirectly through a stub library. This library decouples the extension from a specific version of Tcl.

Example using catch

This proc closes everything wrapped in `catch` with added output if the routine does not return `SQL_SUCCESS*`.

```
#####
# Name:   CloseAfterError
#
#         clean up after an error in a specified part
#         of the database processing
# Usage:  ODBC::ORACLE::CloseAfterError mode [hstmt]
# where:
#         mode - operation being performed when error occurred
#         hstmt - statement handle (optional) #
# Returns: nothing
# Notes:
#         In case of an error during the connect or later ODBC
#         operations, rewind things by doing the free statement handle,
#         disconnect, free connection handle dance. Unset the connection
#         handle namespace variable as the signal that the connection
#         handle should be reallocated, etc., when the next attempt is
#         made to perform a database operation.
#         If and error is encountered in allocating the environment or
#         connection handles, etc., it is probably unrecoverable. An error
#         message is emitted. The process is stopped.
proc CloseAfterError {} {
    variable hdbc
    variable henv
    variable hstmt
    if {$hstmt == {}} {
        puts stderr "ODBC Error - CloseAfterError called without an hstmt parameter set"
        return
    }
    if [ catch { odbc SQLCloseCursor $hstmt } errHstmt ] {
        puts $errHstmt
    }
    set rVal [ catch { odbc SQLFreeStmt $hstmt SQL_DROP } errMsg ] {
        puts stderr "ODBC Error - in SQLFreeStmt SQL_DROP"
    }

    set rVal [ catch { odbc SQLFreeHandle SQL_HANDLE_STMT $hstmt } ]
    if {$rVal != "SQL_SUCCESS" && $rVal != "SQL_SUCCESS_WITH_INFO"} {
        puts stderr "ODBC Error - in SQLFreeHandle SQL_HANDLE_STMT"
    }
    set rVal [ catch { odbc SQLDisconnect $hdbc } ]
    if {$rVal != "SQL_SUCCESS" && $rVal != "SQL_SUCCESS_WITH_INFO"} {
        puts stderr "ODBC Error - in SQLDisconnect $hdbc"
    }
    set rVal [ catch { odbc SQLFreeHandle SQL_HANDLE_DBC $hdbc } ]
    if {$rVal != "SQL_SUCCESS" && $rVal != "SQL_SUCCESS_WITH_INFO"} {
        puts stderr "ODBC Error - in SQLFreeHandle SQL_HANDLE_DBC"
    }
    set hdbc {}
    return
} ;# end close after error
```

ODBC Tcl API and C API differences

The Tcl API is similar to the C API. One set of ODBC documentation can be used for both APIs, and existing ODBC C applications can be ported to Tcl. There are several generic differences between the C ODBC API and these Tcl extensions.

Handles in the C API are pointers to memory. Handles in Tcl are short strings ending in a number, such as environment handle `henv0`, connection handle `hdbc0`, statement handle `hstmt0`, and descriptor handle `hdesc0`.

Some instances in C have an object specified as input. In Tcl, specify a string value through a literal value or with a "\$" preceding a variable name.

For instances in which the C argument is a pointer to an object, the corresponding Tcl argument is a variable name. In most cases the argument is an output argument.

Memory allocation

In C, memory is allocated by the application for input and output buffers. The application passes the address and length of the buffer to a driver.

In some Tcl instances, a buffer must be specified. The argument type is listed in the ODBC documentation as **PTR**. The application passes the name of a Tcl variable and the length. Memory management is handled by Tcl.

In some C instances, the parameter is coded as an array of input or output values. The Tcl counterpart is an array indexed with integers starting at "0."

Data types

With the C API, C data types can be specified as arguments to certain function calls. Tcl objects support a smaller set of data types. For portability, the C coding is accepted, but the result is converted to the closest Tcl datatype, or string, if there is no corresponding datatype.

For example, **SQL_C_LONG**, **SQL_C_SLONG**, and **SQL_C_ULONG** data are converted to Tcl objects of type **Long**. Correspondingly, **SQL_C_DOUBLE** maps to the Tcl **Double** type and **SQL_C_BINARY** maps to the Tcl **ByteArray** type.

Many of the input and output values are specified in terms of named constants which are implemented in C as macros. These names are all supported in Tcl as string constants. The **SQLGetInfo** function documents numerous bitmasks which are used to interpret the results of the function. These bitmasks are implemented in Tcl as global variables and can be used with Tcl's bitwise operators to extract results.

For **SQLDriverConnect**, only the **SQL_DRIVER_PROMPT** value of **fDriverCompletion** is supported.

Error returns

If severe errors happen in the Tcl extension layer unrelated to ODBC, then the Tcl invocation returns **TCL_ERROR**. Otherwise, the Tcl invocation returns **TCL_OK** and sets the interpreter result to the ODBC function return code in string form. For example, **SQL_SUCCESS**. When the situation warrants, the Tcl application can then invoke the **SQLGetDiagRec** function to retrieve error information as a C application would.

Coding examples

These examples explain various coding calls to the Tcl ODBC extensions, such as:

- Loading the ODBC Tcl extensions
- Setting the environment
- Connecting to a data source
- Submitting requests
- Terminating a statement and connection

Other examples are also included.

Dynamically loading the ODBC Tcl extension

The syntax for dynamically loading the ODBC Tcl Extension is:

```
% setroot
% hcitcl
% package require odbc 8.0 (corresponds with the Connect driver)
% exit
```

Setting the environment and connection handles

SQLAllocHandle is called (once) with a handle type of SQL_HANDLE_ENV to initialize the ODBC environment. The value of the Tcl variable, passed as an argument, is set on return from the function to the Tcl environment handle. For example, "henv0."

```
odbc SQLAllocHandle SQL_HANDLE_ENV SQL_NULL_HANDLE \
henv
```

SQLSetEnvAttr is then typically called to declare the application's ODBC version. SQL_OV_ODBC3 is the most current.

```
odbc SQLSetEnvAttr $henv SQL_ATTR_ODBC_VERSION \
SQL_OV_ODBC3 0
```

SQLAllocHandle is finally called to allocate memory for a connection handle. The value of the Tcl variable passed as the last argument is set on return to the Tcl connection handle.

```
odbc SQLAllocHandle SQL_HANDLE_DBC $henv hdbc
```

Connecting to a data source

After environment and connection handles have been allocated, the application can connect to the data source using one of these subfunctions:

- SQLConnect

This passes a connection handle, a data source name, a login ID or account name, and an authentication string, or password. The lengths of the strings are also passed. Depending on the data source, the login ID and the password may be optional. SQL_NTS is used to indicate that the corresponding parameter is a null-terminated character string.

```
odbc SQLConnect $hdbc MSSQLServer SQL_NTS golan \
SQL_NTS golan SQL_NTS
```

- SQLDriverConnect

This connects to data sources which require more information than the arguments supported by SQLConnect. Dialog boxes are not supported for this function. The connection string argument is used to pass all necessary data source-specific connection information.

```
odbc SQLDriverConnect $hdbc NULL \
"DSN=MSSQLServer;UID=golan;PWD=golan" SQL_NTS \
OutConnectionString 128 StringLengthPtr \
SQL_DRIVER_NOPROMPT
```

- SQLBrowseConnect

This determines, in iterative fashion, the necessary information that is required for connection establishment. It must be called a minimum of two times to establish a connection. After a successful connection is established, the connection string it returns can be used with SQLDriverConnect for subsequent connection to the same data source.

```
odbc SQLBrowseConnect $hdbc \
"DSN=MSSQLServer;UID=golan;PWD=golan; \
ADDRESS=odbcetest" SQL_NTS OutConnectionString \
512 StringLength2Ptr
```

Obtaining information about a driver and data source

These functions give information about a driver and data source:

- SQLDataSources

This returns the data source name and description for each data source in the system information.

```
odbc SQLDataSources $henv SQL_FETCH_NEXT ServerName \
128 NameLength1Ptr Description 128 NameLength2Ptr
```

- SQLDrivers

This returns the description and a list of attributes (keyword/value pairs) for a driver in the installed odbccinst file.

```
odbc SQLDrivers $henv SQL_FETCH_NEXT DriverDescription \
128 DescriptionLengthPtr DriverAttributes 128 \
AttributesLengthPtr
```

- SQLGetInfo

This returns an item of information regarding the driver or data source that is associated with a connection handle. There are several values which can be specified *forInfoType*; SQL_SEARCH_PATTERN_ESCAPE is an example.

```
odbc SQLGetInfo $hdbc SQL_SEARCH_PATTERN_ESCAPE \
    InfoValuePtr 255 StringLengthPtr
```

- SQLGetFunctions

This reports whether a driver supports a particular ODBC function. The *SupportedPtr* Tcl variable is set to TRUE or FALSE upon function return. Capitalize the function name and precede it with SQL_API_.

Alternatively, use SQL_API_ALL_FUNCTIONS to get a report for all functions. In this case, the *SupportedPtr* Tcl variable is set to a list in which each list item is a sublist. This lists consists of the SQL_API_ string for a particular value followed by TRUE or FALSE.

```
"{SQL_API_SQLALLOCONNECT TRUE} \
 {SQL_API_SQLALLOCENV TRUE} ... \
 {SQL_API_SQLBINDPARAMETER TRUE}"
odbc SQLGetFunctions $hdbc SQL_API_SQLGETFUNCTIONS \
    SupportedPtr
odbc SQLGetFunctions $hdbc SQL_API_ALL_FUNCTIONS \
    SupportedPt
```

- SQLGetTypeInfo

This returns information about a given data source's support for a given data type. This is the first function which returns its results as a "result set". This is opposed to setting the values of Tcl variables specified as function arguments.

Allocate a statement handle with SQLAllocHandle (see "Preparing SQL requests" in this topic), as result sets are associated with a particular statement handle. The procedures for retrieving results are described in "Submitting requests" in this topic.

If the data type is not supported, then the SQLGetTypeInfo invocation succeeds, but the returned result set is empty.

```
odbc SQLGetTypeInfo $hstmt SQL_INTEGER
odbc SQLGetTypeInfo $hstmt SQL_FLOAT
```

Setting and retrieving driver options

These functions set and retrieve driver options:

- SQLSetConnectAttr

This sets an attribute which governs a connection aspect.

```
odbc SQLSetConnectAttr $hdbc SQL_ATTR_ACCESS_MODE \
    SQL_MODE_READ_ONLY 0
```

- SQLGetConnectAttr

This retrieves the value of a particular connection option.

```
odbc SQLGetConnectAttr $hdbc SQL_ATTR_ACCESS_MODE \
    ValuePtr 0 NULL
```

- SQLSetStmtAttr

This sets an attribute which governs operations that are associated with a particular statement handle.

```
odbc SQLSetStmtAttr $hstmt SQL_ATTR_ASYNC_ENABLE \
    SQL_ASYNC_ENABLE_ON 0
```

- SQLGetStmtAttr

This retrieves the value of a particular statement handle option.

```
odbc SQLGetStmtAttr $hstmt SQL_ATTR_ASYNC_ENABLE \
    ValuePtr 0 NULL
```

Preparing SQL requests

These functions prepare SQL requests:

- SQLAllocHandle

This allocates memory for statement handle called with `SQL_HANDLE_STMT`. The value of the Tcl variable passed as the last argument is set on return to the Tcl statement handle.

```
odbc SQLAllocHandle SQL_HANDLE_STMT $hdbc hstmt
```

- SQLPrepare

This runs a SQL statement multiple times. The statement can contain one or more parameter markers, such as embedded question marks [?].

```
odbc SQLPrepare $hstmt $insert SQL_NTS
```

- SQLBindParameter

This binds a buffer to a parameter marker in a SQL statement.

For "data at statement running" parameters, the C `SQL_LEN_DATA_AT_EXEC` (length) macro has been supported in Tcl as a command. It may be used to supply the `StrLen_or_IndPtr` argument to `SQLBindParameter`.

In this case, the Tcl interface requires the **ParameterValuePtr** argument to be specified as a number. Typically, this is the parameter number.

```
set insert "INSERT INTO NAMEID VALUES (?, ?, ?)"
odbc SQLBindParameter $hstmt 1 SQL_PARAM_INPUT \
    SQL_C_SLONG SQL_INTEGER 0 0 id 0 NULL
odbc SQLBindParameter $hstmt 1 SQL_PARAM_INPUT \
    SQL_C_SLONG SQL_INTEGER 0 0 1 0 \ StrLen_or_IndPtr1
set StrLen_or_IndPtr1 [SQL_LEN_DATA_AT_EXEC 0]
odbc SQLBindParameter $hstmt 2 SQL_PARAM_INPUT \
```

```
SQL_C_DOUBLE SQL_DOUBLE 0 0 dTag 0 NULL
odbc SQLBindParameter $hstmt 3 SQL_PARAM_INPUT \
SQL_C_CHAR SQL_CHAR 50 0 name 50 NULL
```

- `SQLSetStmtAttr`

This sets an attribute that permits multiple values for input parameters. This permits bulk inserts. The Tcl variable bound with `SQLBindParameter` is expected to be a Tcl array with indexes "0," "1," "2," and so on.

```
odbc SQLSetStmtAttr $hstmt SQL_ATTR_PARAMSET_SIZE 7 0
```

- `SQLSetCursorName`

This sets the positioned update and delete. In positioned update/delete, a result set is retrieved and searched for a particular record. It is usually retrieved with a SELECT statement containing a "FOR UPDATE" clause. That record can be modified or deleted using the cursor name assigned with `SQLSetCursorName`. It can also be modified by retrieving the implicit cursor name generated by the driver with `SQLGetCursorName`.

```
odbc SQLSetCursorName $hstmt MyCursor 8
odbc SQLGetCursorName $hstmt CursorName 12 \
NameLengthPtr
```

- `SQLSetStmtAttr`

This sets a multitude of statement attributes including cursor characteristics.

```
odbc SQLSetStmtAttr $hstmt \
SQL_ATTR_CURSOR_SCROLLABLE SQL_SCROLLABLE 0
```

Submitting requests

These functions submit requests:

- `SQLExecute`

This runs the prepared statements.

```
odbc SQLExecute $hstmt
```

- `SQLExecDirect`

This prepares and runs a statement. This is the fastest method for running one instance.

```
set drop "DROP TABLE NAMEID"
odbc SQLExecDirect $hstmt $drop SQL_NTS
```

- `SQLNativeSql`

This shows the translated SQL string as it would be passed to the data source.

```
odbc SQLNativeSql $hdbc "SELECT * FROM MYTABLE" \
    SQL_NTS OutStatementText 128 TextLength2Ptr
```

- SQLDescribeParam

This supplies four parameters describing a parameter marker in a prepared SQL statement.

```
odbc SQLDescribeParam $hstmt 1 DataTypePtr \
    ParameterSizePtr DecimalDigitsPtr NullablePtr
```

- SQLNumParams

This supplies the number of parameters in a statement.

```
odbc SQLNumParams $hstmt ParameterCountPtr
```

- SQLParamData

This retrieves the value of the **ParameterValuePtr** parameter. This parameter was previously set on an invocation to **SQLBindParameter** for a parameter that is specified at statement running time. This is usually the parameter number.

```
odbc SQLParamData $hstmt ValuePtrPtr
```

- SQLPutData

This provides the "data at running time" data.

```
odbc SQLPutData $hstmt id 0
```

Retrieving results and information about results

These functions retrieve results and their information:

- SQLRowCount

This retrieves the number of rows that are affected by an insert, update, or delete request.

```
odbc SQLRowCount $hstmt RowCountPtr
```

- SQLNumResultCols

This retrieves the number of columns in a result set.

```
odbc SQLNumResultCols $hstmt ColumnCountPtr
```

- SQLDescribeCol

This retrieves five parameters describing a column in a result set.

```
odbc SQLDescribeCol $hstmt 1 ColumnName 32 \
    NameLengthPtr DataTypePtr ColumnSizePtr \
    DecimalDigitsPtr NullablePtr
```

- SQLColAttribute

This retrieves descriptor information for a column in a result set.

```
odbc SQLColAttribute $hstmt 1 \
    SQL_DESC_CASE_SENSITIVE CharacterAttributePtr \
    16 StringLengthPtr
NumericAttributePtr
```

- SQLBindCol

This assigns storage for a column in a result set.

```
odbc SQLBindCol $hstmt 1 SQL_C_SLONG id 0 idlen
odbc SQLBindCol $hstmt 2 SQL_C_DOUBLE dTagOut 0 dTaglen
odbc SQLBindCol $hstmt 3 SQL_C_CHAR nameOut 32 namelen
```

- SQLFetch

This retrieves a row of data from a result set.

```
odbc SQLFetch $hstmt
```

- SQLFetchScroll

This retrieves a rowset of data and absolute, relative, or bookmarked position in the rowset.

```
odbc SQLFetchScroll $hstmt SQL_FETCH_NEXT 0
```

If the statement attributes are set before an invocation to `SQLFetch` or `SQLFetchScroll` during the retrieval, then **rgfRowStatus** is set to a Tcl array. This array is indexed starting from "0" with the number of elements equal to the number of rows in the rowset. This number is specified with the **SQL_ROW_ARRAY_SIZE** attribute of `SQLSetStmtAttr`.

```
odbc SQLSetStmtAttr $hstmt SQL_ATTR_ROW_STATUS_PTR \
    rgfRowStatus 0
```

- SQLGetData

This retrieves data from a single unbound column in the current row. The **TargetType** parameter can be coded as for **SQLBindCol**.

```
odbc SQLGetData $hstmt 1 SQL_C_CHAR nameOut 50 \
    StrLen_or_IndPtr1
odbc SQLGetData $hstmt 2 SQL_C_SHORT dTagOut 15 \
    StrLen_or_IndPtr2
odbc SQLGetData $hstmt 3 SQL_C_CHAR idOut 10 \
    StrLen_or_IndPtr3
odbc SQLSetPos $hstmt 4 SQL_POSITION \
    SQL_LOCK_NO_CHANGE
odbc SQLSetPos $hstmt 0 SQL_REFRESH \
    SQL_LOCK_NO_CHANGE
```

```
odbc SQLSetPos $hstmt 2 SQL_UPDATE \
    SQL_LOCK_NO_CHANGE
odbc SQLSetPos $hstmt 3 SQL_DELETE \
    SQL_LOCK_NO_CHANGE
odbc SQLSetPos $hstmt 2 SQL_ADD SQL_LOCK_NO_CHANGE
```

- `SQLMoreResults`

This retrieves multiple result sets.

```
odbc SQLMoreResults $hstmt
```

- `SQLGetDiagRec` and `SQLGetDiagField`

This retrieves ODBC status information.

These examples show how to get information on a particular error that is related to a connection handle or a statement handle, respectively.

```
odbc SQLGetDiagRec SQL_HANDLE_DBC $hdbc 1 SqlState \
    NativeError MessageText 511 TextLength
odbc SQLGetDiagRec SQL_HANDLE_STMT $hstmt 1 \
    SqlState NativeError MessageText 511 TextLength
```

This example shows how to get the field value of an error that is related to a connection handle.

```
odbc SQLGetDiagField SQL_HANDLE_DBC $hdbc 1 \
    SQL_DIAG_SERVER_NAME DiagInfoPtr 256 \
    StringLengthPtr
```

Using descriptors

Use descriptor functions to streamline operations.

```
odbc SQLGetStmtAttr $hstmt SQL_ATTR_APP_ROW_DESC \
    hdesc 32 StringLength
odbc SQLGetDescField $hdesc 0 SQL_DESC_ALLOC_TYPE \
    Value 512 StringLength
odbc SQLGetDescRec $hdesc $RecNumber Name 512 \
    StringLength Type SubType Length Precision Scale Nullable
odbc SQLCopyDesc $hdesc1 $hdesc2
odbc SQLCopyDesc $hdesc1 $hdesc2
```

Terminating a statement

These functions terminate a statement:

- `SQLFreeStmt`

Provides these options for starting over:

```
odbc SQLFreeStmt $hstmt SQL_DROP
odbc SQLFreeStmt $hstmt SQL_RESET_PARAMS
odbc SQLFreeStmt $hstmt SQL_UNBIND
```

- `SQLFreeHandle`

Frees resources that are associated with a statement handle.

```
odbc SQLFreeHandle SQL_HANDLE_STMT $hstmt
```

- `SQLCancel`

Cancels processing on a statement handle.

```
odbc SQLCancel $hstmt
```

- `SQLEndTran`

Attempts to roll back all active operations on all statements that are associated with the given connection:

```
odbc SQLEndTran SQL_HANDLE_DBC $hdbc SQL_ROLLBACK
```

This attempts to commit transactions on all connection handles associated with the given environment:

```
odbc SQLEndTran SQL_HANDLE_ENV $henv SQL_COMMIT
```

Terminating a connection

There are no surprises in these calls to `SQLDisconnect` and `SQLFreeHandle`:

```
odbc SQLDisconnect $hdbc
odbc SQLFreeHandle SQL_HANDLE_DBC $hdbc
odbc SQLFreeHandle SQL_HANDLE_ENV $henv
```

Obtaining information about the data source's system tables

These data dictionary or catalog functions return information about how data is stored in the database:

- `SQLColumnPrivileges`

```
odbc SQLColumnPrivileges $hstmt NULL 0 NULL 0 \
NAMEID SQL_NTS NULL 0
```

- `SQLColumns`

```
odbc SQLColumns $hstmt NULL 0 NULL 0 NAMEID \
SQL_NTS NULL 0
```

- `SQLForeignKeys`

```
odbc SQLForeignKeys $hstmt NULL 0 NULL 0 NAMEID \
SQL_NTS NULL 0 NULL 0 NULL 0
```

- SQLPrimaryKeys

```
odbc SQLPrimaryKeys $hstmt NULL 0 NULL 0 NAMEID \
SQL_NTS
```

- SQLProcedureColumns

```
odbc SQLProcedureColumns $hstmt NULL 0 NULL 0 \
NULL 0 NULL 0
```

- SQLProcedures

```
odbc SQLProcedures $hstmt NULL 0 NULL 0 NULL 0
```

- SQLSpecialColumns

```
odbc SQLSpecialColumns $hstmt SQL_ROWVER NULL 0 \
NULL 0 sysalternates SQL_NTS SQL_SCOPE_SESSION \
SQL_NULLABLE
```

- SQLStatistics

```
odbc SQLStatistics $hstmt NULL 0 NULL 0 NAMEID \
SQL_NTS SQL_INDEX_ALL SQL_ENSURE
```

- SQLTablePrivileges

```
odbc SQLTablePrivileges $hstmt NULL 0 NULL 0 NULL 0
```

- SQLTables

```
odbc SQLTables $hstmt NULL 0 NULL 0 "Orders" \
SQL_NTS NULL 0
```

Application example

This shows an example TPS script using ODBC to modify data in the database through the system. The code shows two methods of inserting data into tables.

This procedure is configured in the **UPoC Protocol Properties** dialog box's **Read TPS** field.

The thread wakes up periodically (the interval in seconds is set in **Read Interval**). Then, it queries a database table called PATIENTS that is being used to place messages into the system.

This procedure:

- Reads all of the rows in the PATIENTS table.
- Generates a message for each row that is retrieved.

- Translates a numeric field in each retrieved row to a string value by performing a lookup in a second table called DOCTORS.
- Formats a system message with the queried information and passes a message into the system.
- Deletes the row from the database.

For this example, start mode code populates the PATIENTS and DOCTORS tables with sample data.

```
#####
# Name: odbcl
# Purpose: <description>
# UPoC type: tps
# Args: tps keyedlist containing the following keys:
#         MODE run mode ("start", "run", "time", or "test")
#         MSGID message handle
#         ARGS user-supplied arguments:
#               <describe user-supplied args here>
#
# Returns: tps disposition list:
#         <describe dispositions used here>
#
proc odbcl { args } {
    keylget args MODE mode;
    # Fetch mode
    switch -exact -- $mode {
        start {
            # Perform special init functions
            # A Microsoft SQL Server database user on pilsner named "med_adm"
            # has been created. His database password is "aspirin". Database
            # "masterpatient" has also been created. The med_adm user has been
            # given all necessary permissions on "masterpatient".
            # Using the following hcitcl script we create a couple of tables in
            # the masterpatient database (named DOCTORS and PATIENTS) and
            # populate them.
            # initialize the ODBC call level interface

            package require odbc
            odbc SQLAllocHandle SQL_HANDLE_ENV SQL_NULL_HANDLE henv

            # application is ODBC 3
            odbc SQLSetEnvAttr $henv SQL_ATTR_ODBC_VERSION SQL_OV_ODBC3 0
            # allocate a connection handle
            odbc SQLAllocHandle SQL_HANDLE_DBC $henv hdbc

            # connect to the data source: data source name: MSSQLServer
            # user name: med_adm password: aspirin
            odbc SQLConnect $hdbc MSSQLServer SQL_NTS med_adm SQL_NTS \
                aspirin SQL_NTS

            # allocate a statement handle
            odbc SQLAllocHandle SQL_HANDLE_STMT $hdbc hstmt

            # drop the DOCTORS table, if it exists
            # the call will return SQL_ERROR if it fails, but no worries
            set drop "DROP TABLE DOCTORS"
            odbc SQLExecDirect $hstmt $drop SQL_NTS

            # recreate the table
            set create "CREATE TABLE DOCTORS (ID INT, NAME VARCHAR(32))"
            odbc SQLExecDirect $hstmt $create SQL_NTS

            # prepare an insert statement with 2 parameter markers
            set insert "INSERT INTO DOCTORS VALUES (?, ?)"
            odbc SQLPrepare $hstmt $insert SQL_NTS

            # bind tcl variables to these two parameters
            odbc SQLBindParameter $hstmt 1 SQL_PARAM_INPUT SQL_C_SLONG \
                SQL_INTEGER 0 0 id 0 NULL
            odbc SQLBindParameter $hstmt 2 SQL_PARAM_INPUT SQL_C_CHAR \
                SQL_CHAR 32 0 name 32 NULL
        }
    }
}
```

```

# insert several rows into the database,
# repeatedly executing the prepared statement
    set id 100; set name "Frankenstein"; odbc SQLExecute $hstmt
    set id 101; set name "Kildare"; odbc SQLExecute $hstmt
    set id 102; set name "Jekyll"; odbc SQLExecute $hstmt
    set id 103; set name "Kevorkian"; odbc SQLExecute $hstmt
    set id 104; set name "Spock"; odbc SQLExecute $hstmt
    set id 105; set name "Watson"; odbc SQLExecute $hstmt
    set id 106; set name "Ruth"; odbc SQLExecute $hstmt
    unset id name

# drop another existing table
    set drop "DROP TABLE PATIENTS"
    odbc SQLExecDirect $hstmt $drop SQL_NTS

# recreate the table
    set create "CREATE TABLE PATIENTS (ID INT, NAME \
        VARCHAR(32),DRID INT)"
    odbc SQLExecDirect $hstmt $create SQL_NTS

# prepare an insert statement with 3 parameter markers
    set insert "INSERT INTO PATIENTS VALUES (?, ?, ?)"
    odbc SQLPrepare $hstmt $insert SQL_NTS

# we will now show a second method of inserting values
# into a table which only requires a single SQLExecute
# set up 2 Tcl arrays with the input values for the insert
    set id(0) 600; set name(0) "Mantle"; set drid(0) 106;
    set id(1) 601; set name(1) "Dimaggio"; set drid(1) 105;
    set id(2) 602; set name(2) "Ruth"; set drid(2) 104;
    set id(3) 603; set name(3) "Gehrig"; set drid(3) 103;
    set id(4) 604; set name(4) "Stengel"; set drid(4) 102;
    set id(5) 605; set name(5) "Berra"; set drid(5) 101;
    set id(6) 606; set name(6) "Jackson"; set drid(6) 100;

# set things up for multiple parameter values
    odbc SQLSetStmtAttr $hstmt SQL_ATTR_PARAMSET_SIZE 7 0

# perform the bulk insert; 7 rows are inserted with one ODBC call
# the parameter bindings are still in effect
    odbc SQLBindParameter $hstmt 3 SQL_PARAM_INPUT SQL_C_SLONG \
        SQL_INTEGER 0 0 drid 0 NULL
    odbc SQLExecute $hstmt
    unset id name drid
    odbc SQLFreeHandle SQL_HANDLE_STMT $hstmt
    odbc SQLDisconnect $hdbc
    odbc SQLFreeHandle SQL_HANDLE_DBC $hdbc
    odbc SQLFreeHandle SQL_HANDLE_ENV $henv
    return
}

run {
    # "run" mode always has a MSGID; fetch and process it
}

time {
    # Timer-based processing

    odbc SQLAllocHandle SQL_HANDLE_ENV SQL_NULL_HANDLE henv
    odbc SQLSetEnvAttr $henv SQL_ATTR_ODBC_VERSION SQL_OV_ODBC3 0
# allocate a connection handle
    odbc SQLAllocHandle SQL_HANDLE_DBC $henv hdbc
    odbc SQLConnect $hdbc MSSQLServer SQL_NTS med_adm \
        SQL_NTS aspirin SQL_NTS
    odbc SQLAllocHandle SQL_HANDLE_STMT $hdbc hstmt

# the following code is for the case where incoming messages are
# being deposited into the PATIENT database which is being
# periodically queried by Tcl code in a UPOC driver

# a result set cursor defaults to read-only. the following enables #locking for
# safe updates and deletes
    odbc SQLSetStmtAttr $hstmt SQL_ATTR_CONCURRENCY \
        SQL_CONCUR_LOCK 0

```

```

# the "FOR UPDATE" clause permits the use of a named cursor
# for positioned result set operations
set select "SELECT * FROM PATIENTS FOR UPDATE"
odbc SQLExecDirect $hstmt $select SQL_NTS

# we now have a result set and a default cursor name
# we could name our own cursor with SQLSetCursorName
# but we just use the default
odbc SQLGetCursorName $hstmt cursorName 20 pcbCursor

# we need a second statement handle to do a positioned delete
# to get rid of the record in the database after we have read it.
odbc SQLAllocHandle SQL_HANDLE_STMT $hdbc hstmt2

# prepare the positioned delete using the default cursor name
# returned from the SQLGetCursorName query.
set delete "DELETE FROM PATIENTS WHERE CURRENT OF $cursorName"
odbc SQLPrepare $hstmt2 $delete SQL_NTS

# bind the two columns in the result set from the previous
# "select *"
odbc SQLBindCol $hstmt 1 SQL_C_CHAR id 10 pcbValue1
odbc SQLBindCol $hstmt 2 SQL_C_CHAR name 32 pcbValue2
odbc SQLBindCol $hstmt 3 SQL_C_CHAR drid 10 pcbValue3

# we need a third statement handle to look up the name of the
# doctor assigned to the patient
odbc SQLAllocHandle SQL_HANDLE_STMT $hdbc hstmt3
odbc SQLBindCol $hstmt3 1 SQL_C_CHAR drname 32 pcbValueDr

# now loop over the result set fetching out each row's values and
# then deleting the row from the PATIENTS table in the database
set dispList {}
set rVal [odbc SQLFetch $hstmt]
while {$rVal == "SQL_SUCCESS" || $rVal == \
  "SQL_SUCCESS_WITH_INFO"} {
  set select "SELECT NAME FROM DOCTORS WHERE ID = $drid"
  odbc SQLExecDirect $hstmt3 $select SQL_NTS
  odbc SQLFetch $hstmt3
  odbc SQLFreeStmt $hstmt3 SQL_DROP
# positioned delete
set rVal2 [odbc SQLExecute $hstmt2]
echo "return code from delete:$rVal2"
set mh [msgcreate "patient identifier: $id patient name: \
  $name doctor: $drname"]
lappend dispList "CONTINUE $mh"
set rVal [odbc SQLFetch $hstmt]
}

# now we've read and deleted all of the records
odbc SQLFreeHandle SQL_HANDLE_STMT $hstmt
odbc SQLFreeHandle SQL_HANDLE_STMT $hstmt2
odbc SQLFreeHandle SQL_HANDLE_STMT $hstmt3
odbc SQLDisconnect $hdbc
odbc SQLFreeHandle SQL_HANDLE_DBC $hdbc
odbc SQLFreeHandle SQL_HANDLE_ENV $henv
return $dispList
}
default {
  error "Unknown mode '$mode' in odbc1"
}
}
}

```

ODBC command reference

The `odbc` command provides access from within Tcl to the ODBC API (Open DataBase Connectivity Application Programming Interface). The function of each ODBC API is denoted under the command column. Functions may also be specified in mixed case, as they are in the ODBC API documentation. For example, `InputHandle` instead of than `inputhandle`.

For a list of ODBC commands with their options and description, see [ODBC commands](#).

For a list of deprecated ODBC commands with their options and description, see [Deprecated ODBC commands](#). These commands are still currently supported, but support could be withdrawn in future releases.

Note: Each command corresponds to an ODBC API function of the same name, without the ODBC prefix.

ODBC commands

This table lists the ODBC commands:

Command	Options	Description
<code>odbc sqlallochandle</code>	<ul style="list-style-type: none"> <code>HandleType</code> <code>InputHandle</code> <code>OutputHandlePtr</code> 	Allocates memory for one of the four types of ODBC handles: environment, connection, statement, or descriptor.
<code>odbc sqlbindcol</code>	<ul style="list-style-type: none"> <code>hstmt</code> <code>icol</code> <code>fCType</code> <code>rgbValue</code> <code>cbValueMax</code> <code>pcbValue</code> 	Assigns the storage and data type for a column in a result set.
<code>odbc sqlbindparameter</code>	<ul style="list-style-type: none"> <code>hstmt</code> <code>ipar</code> <code>fParamType</code> <code>fCType</code> <code>fSqlType</code> <code>cbColDef</code> <code>ibScale</code> <code>rgbValue</code> <code>cbValueMax</code> <code>pcbValue</code> 	Binds a buffer to a parameter marker in an SQL statement.

Command	Options	Description
odbc sqlbrowseconnect	<ul style="list-style-type: none"> • hdbc • szConnStrIn • cbConnStrIn • szConnStrOut • cbConnStrOutMax • pcbConnStrOut 	Supports an iterative method of discovering and enumerating the attributes and attribute values required to connect to a data source.
odbc sqlbulkoperations	<ul style="list-style-type: none"> • StatementHandle • Operation 	Performs bulk insertions and bulk bookmark operations: update, delete, retrieve by bookmark.
odbc sqlcancel	hstmt	Cancels the processing on a statement.
odbc sqlclosecursor	StatementHandle	Closes a cursor that has been opened on a statement and discards pending results.
odbc sqlcolattribute	<ul style="list-style-type: none"> • StatementHandle • ColumnNumber • FieldIdentifier • CharacterAttributePtr • BufferLength • StringLengthPtr • NumericAttributePtr 	Returns descriptor information for a column in a result set.
odbc sqlcolumnprivileges	<ul style="list-style-type: none"> • hstmt • szTableQualifier • cbTableQualifier • szTableOwner • cbTableOwner • szTableName • cbTableName • szColumnName • cbColumnName 	Supplies a list of columns and associated privileges for the specified table.
odbc sqlcolumns	<ul style="list-style-type: none"> • hstmt • szTableQualifier • cbTableQualifier • szTableOwner • cbTableOwner • szTableName • cbTableName • szColumnName • cbColumnName 	Supplies a list of column names in specified tables.

Command	Options	Description
odbc sqlconnect	<ul style="list-style-type: none"> • hdbc • szDSN • cbDSN • szUID • cbUID • szAuthStr • cbAuthStr [-encoding encName] 	Loads a driver and establishes a connection to a data source.
odbc sqlcopydesc	<ul style="list-style-type: none"> • SourceDescHandle • TargetDescHandle 	Copies descriptor information from one descriptor handle to another.
odbc sqldatasources	<ul style="list-style-type: none"> • henv • fDirection • szDSN • cbDSNMax • pcbDSN • szDescription • cbDescriptionMax • pcbDescription 	Lists data source names.
odbc sqldescribecol	<ul style="list-style-type: none"> • hstmt • icol • szColName • cbColNameMax • pcbColName • pfSqlType • pcbColDef • pibScale • pfNullable 	Supplies a description of one column in the result set.
odbc sqldescribeparam	<ul style="list-style-type: none"> • hstmt • ipar • pfSqlType • pcbColDef • pibScale • pfNullable 	Supplies the description of a parameter marker that is associated with a prepared SQL statement.
odbc sqldisconnect	hdbc	Closes the connection that is associated with a specific connection handle.

Command	Options	Description
odbc sqldriverconnect	<ul style="list-style-type: none"> hdbc hwnd szConnStrIn cbConnStrIn szConnStrOut cbConnStrOutMax pcbConnStrOut fDriverCompletion 	An alternative to SQLConnect for connecting to a data source. Use if more connection information than the three arguments in SQLConnect are required.
odbc sqldrivers	<ul style="list-style-type: none"> henv fDirection szDriverDesc cbDriverDescMax pcbDriverDesc szDriverAttributes cbDrvAttrMax pcbDrvAttr 	Lists driver descriptions and driver attribute keywords.
odbc sqlendtran	<ul style="list-style-type: none"> HandleType Handle CompletionType 	Requests a commit or rollback operation for active operations on statements that are associated with a connection or all connections associated with an environment.
odbc sqlexecdirect	<ul style="list-style-type: none"> hstmt szSqlStr cbSqlStr 	Runs a preparable statement using the current values of the parameter marker variables, if any parameters exist in the statement. This is the fastest way to submit an SQL statement for running one instance.
odbc sqlexecute	hstmt	Runs a prepared statement, using the current values of the parameter marker variables, if any parameter markers exist in the statement.
odbc sqlfetch	hstmt	Retrieves a row of data from a result set. The driver provides data for all columns that were bound to storage locations with SQLBindCol.
odbc sqlfetchscroll	<ul style="list-style-type: none"> StatementHandle FetchOrientation FetchOffset 	Retrieves the specified rowset of data from the result set and returns data for all bound columns.

Command	Options	Description
odbc sqlforeignkeys	<ul style="list-style-type: none"> hstmt szPkTableQualifier cbPkTableQualifier szPkTableOwner cbPkTableOwner szPkTableName cbPkTableName szFkTableQualifier cbFkTableQualifier szPkTableOwner cbPkTableOwner szPkTableName cbPkTableName 	<p>This supplies:</p> <ul style="list-style-type: none"> A list of foreign keys in the specified table. A list of foreign keys in other tables that refer to the primary key in the specified table.
odbc sqlfreehandle	<ul style="list-style-type: none"> HandleType Handle 	Frees resources that are associated with a specific environment, connection, statement, or descriptor handle.
odbc sqlfreestmt	<ul style="list-style-type: none"> hstmt free_option, where free_option is SQL_UNBIND, SQL_RESET_PARAMS, OR SQL_DROP 	Stops processing that is associated with a specific statement. It also closes any open cursors associated with hstmt, discards pending results, and, optionally, frees all resources associated with the statement handle.
odbc sqlgetconnectattr	<ul style="list-style-type: none"> ConnectionHandle Attribute ValuePtr BufferLength StringLengthPtr 	Returns the current setting of a connection attribute.
odbc sqlgetcursorname	<ul style="list-style-type: none"> hstmt szCursor cbCursorMax pcbCursor 	Supplies the cursor name that is associated with a specific statement.
odbc sqlgetdata	<ul style="list-style-type: none"> hstmt icol fCType rgbValue cbValueMax pcbValue 	Provides result data for a single unbound column in the current row.

Command	Options	Description
odbc sqlgetdescfield	<ul style="list-style-type: none"> DescriptorHandle RecNumber FieldIdentifier ValuePtr BufferLength StringLengthPtr 	Returns the current setting or value of a single field of a descriptor record.
odbc sqlgetdescrec	<ul style="list-style-type: none"> DescriptorHandle RecNumber Name BufferLength StringLengthPtr TypePtr SubTypePtr LengthPtr PrecisionPtr ScalePtr NullablePtr 	Returns the current setting or values of multiple fields of a descriptor record.
odbc sqlgetdiagfield	<ul style="list-style-type: none"> HandleType Handle RecNumber DiagIdentifier DiagInfoPtr BufferLength StringLengthPtr 	Returns the current value of a field of the diagnostic data structure that is associated with a specific handle containing error, warning, and status information.
odbc sqlgetdiagrec	<ul style="list-style-type: none"> HandleType Handle RecNumber Sqlstate NativeErrorPtr MessageText BufferLength TextLengthPtr 	Returns the current values of multiple fields of a diagnostic record that contains error, warning, and status information.
odbc sqlgetenvattr	<ul style="list-style-type: none"> EnvironmentHandle Attribute ValuePtr BufferLength StringLengthPtr 	Returns the current setting of an environment attribute.
odbc sqlgetfunctions	<ul style="list-style-type: none"> hdbc fFunction pfExists 	Supplies information about whether a driver supports a specific ODBC function.

Command	Options	Description
odbc sqlgetinfo	<ul style="list-style-type: none"> hdbc fInfoType rgbInfoValue cbInfoValueMax pcbInfoValue 	Supplies general information about the driver and data source that are associated with a connection.
odbc sqlgetstmtattr	<ul style="list-style-type: none"> StatementHandle Attribute ValuePtr BufferLength StringLengthPtr 	Returns the current setting of a statement attribute.
odbc sqlgettypeinfo	<ul style="list-style-type: none"> hstmt fSqlType 	Supplies information about data types supported by the data source.
odbc sqlmoreresults	hstmt	Determines whether there are more results available on an <code>hstmt</code> containing SELECT, UPDATE, INSERT, or DELETE statements. If so, then it initializes processing for those results.
odbc sqlnativesq	<ul style="list-style-type: none"> hdbc szSqlStrIn cbSqlStrIn szSqlStr cbSqlStrMax pcbSqlStr 	Supplies the SQL string as translated by the driver.
odbc sqlnumparams	<ul style="list-style-type: none"> hstmt pcpar 	Supplies the number of parameters in an SQL statement.
odbc sqlnumresultcols	<ul style="list-style-type: none"> hstmt pccol 	Supplies the number of columns in a result set.
odbc sqlnumresultcols	<ul style="list-style-type: none"> hstmt pccol 	Supplies the number of columns in a result set.
odbc sqlparamdata	<ul style="list-style-type: none"> hstmt rgbValue 	Used in conjunction with <code>SQLPutData</code> to supply parameter data at statement running time.
odbc sqlprepare	<ul style="list-style-type: none"> hstmt szSqlStr cbSqlStr 	Prepares an SQL string for running.

Command	Options	Description
odbc sqlprimarykeys	<ul style="list-style-type: none"> • hstmt • szTableQualifier • cbTableQualifier • szTableOwner • cbTableOwner • szTableName • cbTableName 	Returns the column names that makes up the primary key for a table.
odbc sqlprocedurecolumns	<ul style="list-style-type: none"> • hstmt • szProcQualifier • cbProcQualifier • szProcOwner • cbProcOwner • szProcName • cbProcName • szColumnName • cbColumnName 	Returns the list of input and output parameters, including the columns that make up the result set for the specified procedures.
odbc sqlprocedures	<ul style="list-style-type: none"> • hstmt • szProcQualifier • cbProcQualifier • szProcOwner • cbProcOwner • szProcName • cbProcName 	Returns the list of procedure names that are stored in a specific data source.
odbc sqlputdata	<ul style="list-style-type: none"> • hstmt • rgbValue • cbValue 	Sends data for a parameter or column to the driver at statement running time.
odbc sqlrowcount	<ul style="list-style-type: none"> • hstmt • pcrow 	Returns the number of rows that are affected by an UPDATE, INSERT, or DELETE statement or by a SQL_UPDATE, SQL_ADD, or SQL_DELETE operation in SQLSetPos.
odbc sqlsetconnectattr	<ul style="list-style-type: none"> • HandleType • InputHandle • OutputHandlePtr 	Sets attributes that govern aspects of connections.
odbc sqlsetcursorname	<ul style="list-style-type: none"> • hstmt • szCursor • cbCursor 	Associates a cursor name with an active statement.

Command	Options	Description
odbc sqlsetdescfield	<ul style="list-style-type: none"> DescriptorHandle RecNumber FieldIdentifier ValuePtr BufferLength 	Sets the value of a single field of a descriptor record.
odbc sqlsetdescrec	<ul style="list-style-type: none"> DescriptorHandle RecNumber Type SubType Length Precision Scale DataPtr StringLengthPtr IndiDataPtrcatorPtr 	Sets multiple descriptor fields that affect the data type and buffer that are bound to a column or parameter.
odbc sqlsetenvattr	<ul style="list-style-type: none"> EnvironmentHandle Attribute ValuePtr StringLength 	Sets attributes that govern aspects of environments.
odbc sqlsetpos	<ul style="list-style-type: none"> hstmt irow fOption fLock 	Sets the cursor position in a rowset and refreshes, updates, deletes, or adds data to the rowset.
odbc sqlspecialcolumns	<ul style="list-style-type: none"> hstmt fColType szTableQualifier cbTableQualifier szTableOwner cbTableOwner szTableName cbTableName fScope fNullable 	<p>Retrieves this information about columns within a specified table:</p> <ul style="list-style-type: none"> The optimal set of columns that uniquely identifies a row in the table. Columns that are automatically updated when any value in the row is updated by a transaction.
odbc sqlsetstmtattr	<ul style="list-style-type: none"> StatementHandle Attribute ValuePtr StringLength 	Sets attributes that are related to a statement.

Command	Options	Description
odbc sqlstatistics	<ul style="list-style-type: none"> hstmt szTableQualifier cbTableQualifier cbTableName fUnique fAccuracy 	Retrieves a list of statistics about a single table and the indexes that are associated with the table.
odbc sqltableprivileges	<ul style="list-style-type: none"> hstmt szTableQualifier cbTableQualifier szTableOwner cbTableOwner szTableName cbTableName 	Returns a list of tables and the privileges that are associated with each table.
odbc sqltables	<ul style="list-style-type: none"> hstmt szTableQualifier cbTableQualifier szTableOwner cbTableOwner szTableName cbTableName szTableType cbTableType 	Returns the list of table names that are stored in a specific data source.

ODBC deprecated commands

This table lists the ODBC deprecated commands:

Command	Options	Description
odbc sqlallocconnect	<ul style="list-style-type: none"> henv hdbc 	Allocates memory for a connection handle within the specified environment. This also sets the variable specified in hdbc to the new connection handle.

Command	Options	Description
<code>odbc sqlallocenv</code>	<code>henv</code>	Allocates memory for an environment handle and initializes the ODBC API for use by an application. An application must invoke <code>odbc sqlallocenv</code> before invoking any other <code>odbc</code> function. This also sets the variable specified in <code>henv</code> to the new environment handle.
<code>odbc sqlallocstmt</code>	<ul style="list-style-type: none"> <code>hdbc</code> <code>hstmt</code> 	Allocates memory for a statement handle and associates the statement handle with the connection specified by <code>hdbc</code> . This also sets the variable specified in <code>hstmt</code> to the new statement handle.
<code>odbc sqlcolattribute</code>	<ul style="list-style-type: none"> <code>hstmt</code> <code>icol</code> <code>fDescType</code> <code>rgbDesc</code> <code>cbDescMax</code> <code>pcbDesc</code> <code>pfDesc</code> 	Supplies descriptive information for a column in a result set.
<code>odbc sqlerror</code>	<ul style="list-style-type: none"> <code>henv</code> <code>hdbc</code> <code>hstmt</code> <code>szSqlState</code> <code>pfNativeError</code> <code>szErrorMsg</code> <code>cbErrorMsgMax</code> <code>pcbErrorMsg</code> 	Supplies error or status information.
<code>odbc sqlextendedfetch</code>	<ul style="list-style-type: none"> <code>hstmt</code> <code>fFetchType</code> <code>irow</code> <code>pcrow</code> <code>rgfRowStatus</code> 	Extends the functionality of <code>SQLFetch</code> . Retrieves multiple rows and returns them in arrays.
<code>odbc sqlfreeconnect</code>	<code>hdbc</code>	Releases a connection handle and frees all memory that is associated with the handle.

Command	Options	Description
odbc sqlfreeenv	henv	Frees the environment handle and releases all memory that is associated with the environment handle.
odbc sqlgetconnectoption	<ul style="list-style-type: none"> hdbc fOption pvParam 	Supplies the current setting of a connection option.
odbc sqlgetstmtoption	<ul style="list-style-type: none"> hstmt fOption pvParam 	Supplies the current setting of a statement option.
odbc sqlparamoptions	<ul style="list-style-type: none"> hstmt crow pirow 	Specifies multiple values for the set of parameters that are assigned by SQLBindParameter.
odbc sqlsetconnectoption	<ul style="list-style-type: none"> hdbc fOption vParam 	Sets options which govern aspects of connections.
odbc sqlsetparam	<ul style="list-style-type: none"> hstmt ipar fCType fSqlType cbColDef ibScale rgbValue pcbValue 	This ODBC 1.0 function has been replaced by the ODBC 2.0 function SQLBindParameter.
odbc sqlsetscrolloptions	<ul style="list-style-type: none"> hstmt fConcurrency crowKeyset crowRowset 	Sets options that control the behavior of cursors that are associated with an hstmt.
odbc sqlsetstmtoption	<ul style="list-style-type: none"> hstmt fOption vParam 	Sets options that are related to an hstmt.

Command	Options	Description
odbc sqltransact	<ul style="list-style-type: none">henvhdbcfType	<p>Requests a commit or rollback operation for:</p> <ul style="list-style-type: none">All active operations on all <code>hs_tmt</code> that are associated with connection <code>hdbc</code>.orAll active operations on all <code>hs_tmt</code> for all connections in <code>henv</code>.

Troubleshooting

These topics can aid in troubleshooting the error and recovery databases:

- [Error database states](#) on page 1583
- [State 107 note](#) on page 1585
- [State 416 note](#) on page 1586
- [Recovery database states](#) on page 1586
- [Recovery database troubleshooting](#) on page 1587
- [DB Vista -921 recovery](#) on page 1587
- [DB Vista -921 user ID check failure](#) on page 1587
- [DB Vista 940 recovery](#) on page 1588
- [Resetting the database and status](#) on page 1588
- [Resetting the database contents](#) on page 1588
- [Resetting the status](#) on page 1589

Error database states

This table lists all error database states:

State	Description
100	No trxid found
101	Unsupported trxid
103	Tcl failure in trxid determination
104:	No destination for reply message
105	No destination for data message
106	Blocked gateway routing
107	Incorrect remote ICL server (See State 107 note)
200	Tcl failure in IB reply TPS
201	Tcl failure in IB data TPS

State	Description
202	Tcl failure in OB reply TPS
203	Tcl failure in OB data TPS
204	Tcl failure in FWD reply TPS
205	Tcl failure in FWD data TPS
300	XLT config file unloadable
301	Internal XPM Tcl failure
302	Tcl callout error
303	Tcl callout abort
304	XPM data retrieval failed
305	XPM data store failed
306	XPM bulkcopy operation failed
307	Input data validation failure
308	Output data validation failure
309	XPM default value retrieval failed
310	XPM math operation error
311	<code>\$xlateOutVals</code> unset
312	Numeric compare error
313	String compare error
314	XSLT transformation failure
400	Tcl failure in startup TPS
401	Tcl failure in ACK/NAK TPS
402	Tcl failure in <code>SendOK</code> TPS
403	Tcl failure in <code>SendFail</code> TPS
404	Tcl failure in <code>ReplyGen</code> TPS
405	Message undeliverable within retries
406	Tcl failure in startup <code>SendOK</code> TPS
407	Tcl failure in startup <code>SendFail</code> TPS
408	Tcl failure in protocol driver TPS
409	Tcl failure in <code>PreWrite</code> TPS

State	Description
410	Failed to recover reserved outbound message
414	User code error in UPoC protocol read TPS
415	User code error in UPoC protocol write TPS
416	Inbound encoding conversion error (See State 416 note)
417	Outbound encoding conversion error
418	Unable to deliver to specified server connection
419	Bad data in message writing
500	Internal failure: unable to read message data chain
501	Error database TPS error
1000	Internal failure: Start route
1001	Internal failure: Bad route config
1002	Internal failure: No route config
1003	Internal failure: Bad route detail
1004	Internal failure: Bad format

State 107 note

Error database state 107: EDB_ROUTE_REMOTEICLDEST (incorrect remote ICL server). This happens when an inter-site routing message, rdb state 7 RDB_XLATE_POST_XLATE, is being recovered from the start-up of the engine. In some instances, the remote ICL server, or ICL destination, is unresponsive and the ICL destination definition is changed in NetConfig. When this happens, the message is moved to the error database with state 107.

The message is put into the error database during recovery from the recovery database if both of these conditions are met:

- The message cannot be delivered to the port it thinks it should go to.
The message stays in the recovery database and in the queue because the server is unresponsive.
- The configuration, as loaded when the process/thread is started, no longer matches the port the message expects.
The message stays in the recovery database because the configuration has changed, but the message can be delivered. Delivery continues when possible.

During the error database resend, the port is corrected with the newly configured port.

Note: The destination, including port number, is set when a message is routed, not when the message is delivered.

State 416 note

A message with this error state indicates an error has happened during encoding. This error happened during the encoding setting to UTF-8 when it was received in the inbound protocol thread. For example, the encoding setting of the inbound thread does not match the real message. In this situation, the message keeps its original encoding in the SMAT.

When resending such a message from SMAT, the engine converts it from the `-e` option, given by the `resend` command. Conversion is made to the encoding setting of the inbound thread. It then puts the message in the engine queue. In the engine queue, the message is treated as a new message from the inbound thread. The engine does encoding conversion from the encoding setting of the inbound thread to UTF-8 again.

If the resend command does not give the `-e` option, then the engine skips the first encoding conversion. It then puts the message from SMAT in the engine queue.

Recovery database states

This table lists all recovery database states:

State	Description
0	Invalid
1	IB pre-TPS
2	IB post-TPS
3	ICL proto sent
4	ICL xlate staged
5	Pre xlate
6	Partial xlate
7	Post xlate
8	ICL xlate sent
9	ICL proto staged
10	OB pre-TPS
11	OB post-TPS
12	Fwd pre-TPS

State	Description
13	Fwd post-TPS
14	OB delivered
15	OB delivery failed
16	OB wait for IB reply
17	OB prewrite
18	Unbacked queue
19	On java IB protocol
20	On java OB protocol

Recovery database troubleshooting

The error database holds all of the messages that error out during processing. For example, states above 100: no route for TrxId, parse error, and so on.

These topics explain DB Vista errors that can be encountered and how to handle them.

Included are steps to take for resetting the database and status, and destroying the database contents.

DB Vista -921 recovery

This error happens when a DB Vista tool has logged on to the database but crashed before logging off, leaving the user permanently logged on.

Note: Pressing **Ctrl+C** when an `hcidbdump` command is running can cause this error.

To resolve the issue on UNIX, use the `lmc1ear -u TEST -mp` command.

On Windows, stop all processes and daemons. Restart the system from **Control Panel > Administrative Tools > Services**.

DB Vista -921 user ID check failure

This error indicates that a user meets an exception, does not exit normally, and is not cleared when opening or operating a Raima database. When a user attempts to re-open the Raima database, the -921 error is reported.

For example:

```
(-921) 'DB_VISTA user id check failed: '_hcimonitorD_'  
PANIC: "(errnum > -900) || (errnum < -966)"
```

To correct this:

- 1 Run `lmclean -s` to see whether there is a user that did not exit normally.
- 2 Run `lmclean -u $username` to clear up the user.
- 3 Rerun the program in which this error is reported. For example, rerun `monitorD`, `engine`, `hcidbdump`, or other programs.

DB Vista 940 recovery

This error happens when a database file cannot be opened because another application has it open. The application must first close the database file and then the system can start.

For example, these are some applications that could have opened the file, such as a backup or anti-virus program.

These applications must be configured to ignore any site database directories.

Resetting the database and status

- 1 Shut down all processes and the Monitor Daemon.
UNIX: Stop the Lock Manager.
Windows: Stop the CIS service.
 - 2 Run `hcimsiutil -R` to remove the shared memory region.
 - 3 Delete the `rdm.taf` file from the `$HCISITEDIR/exec/databases` directory.
 - 4 Delete `monitorShmemFile` from the `$HCISITEDIR/exec` directory.
 - 5 Initialize ICL by running `hcidbinit -I`.
 - 6 Rebuild the database supporting files by running `hcidbcheck -r`.
 - 7 Restart the Lock Manager, Monitor Daemon, and CIS service.
 - 8 Run `hcidbdump -r`.
- If problems persist, then contact Support.

Resetting the database contents

- 1 Shut down all processes and the Monitor Daemon.
UNIX: Stop the Lock Manager.

Windows: Stop the CIS service.

- 2 Delete the databases directory in `$HCISITEDIR/exec`.
- 3 Copy the databases directory from the "siteProto" site template and paste it into the `$HCISITEDIR/exec` directory.
- 4 Reset these keys in `$HCISITEDIR/siteInfo` to the plain text database:
 - `dbencryption=0`
 - `internaldbkey=`
 - `errordbencryption=0`
 - `errordbkey=`
- 5 Restart the Monitor Daemon and Lock Manager.
- 6 Start the system connections.

Resetting the status

- 1 Shut down all processes and the Monitor Daemon.
- 2 Stop the host server.
- 3 On Windows, stop the CIS service.
- 4 Run `hcimsiutil -R` to remove the shared memory region.
- 5 Delete `monitorShmemFile` from the `$HCISITEDIR/exec` directory.
- 6 Restart the Monitor Daemon and CIS service.
- 7 Start the system connections.

Cloverleaf tips

Tips in configuring and maintaining your Cloverleaf system include:

- [HL7 scripts](#) on page 1590
- [Using xlateInVals and xlateOutVals](#) on page 1600
- [Accessing Tcl help](#) on page 1602
- [Fetching value](#) on page 1602
- [Using Tcl string map](#) on page 1605
- [External message flags bit masks](#) on page 1606
- [Upgrading encodings to MB or 6.2 and later versions](#) on page 1607
- [Configuring a standard inbound TCP/IP server thread](#) on page 1607
- [Configuring a standard outbound TCP/IP client thread](#) on page 1608
- [Using the Fileset protocols without a custom TPS procedure](#) on page 1608
- [Deleting a single message](#) on page 1609
- [Using UltraLong format](#) on page 1609
- [Performing search functions](#) on page 1609
- [Recovery database lock](#) on page 1610
- [Setting up UNIX users](#) on page 1610
- [Adding/removing hciss auto-starting on UNIX/Linux](#) on page 1611
- [Installing the host server as a service in UNIX/Linux](#) on page 1611
- [Recovery database](#) on page 1611

HL7 scripts

HL7 scripts can be used with SMAT .msg files, HL7 files, and the engine from the command line, and are designed to work with pipes.

You can pipe the output to grep, another script, perl, and so on. You can pipe the output of another command or script to one of these scripts. These scripts work on Windows, UNIX, and Linux.

msgExtract

This converts the SMAT .msg file to a newline-delimited file. This is useful when you create a file that is resent to a thread with hci cmd.

```
#!/hci/cloverleaf/cis6.2/integrator/bin/perl
#####
#
# Name: msgExtract
# Purpose: Convert tin/tout file(s) to newline-delimited files (one message per line)
# Note: works on multiple files (ex. msgExtract bn10ms4_in.* > tmp.hl7)
#
#####

#$/ = "";

while (<>) {
    s/MSH\\|/\\nMSH\\|/g;
    s/^\\n//;
    s/\\/\\x0A/;
    print;
}

# End of Script
```

msgExtract examples

Examples of using msgExtract include:

- ```
msgExtract bnd01ms4p_in.tin.msg > test.hl7
```

This converts the whole SMAT .msg file to a newline-delimited file called test.hl7. This is useful for resending the message to an inbound thread. For example:

```
hcicmd -p bnd01 -c 'bnd01ms4p_in resend
ib_pre_tps data 520 test.hl7 nl'
```

- ```
msgExtract
*.tin.msg > test.hl7
```

This converts all .tin.msg files in the current directory to a newline-delimited file called test.hl7.

- ```
msgExtract
bnd01ms4p_in.tin.msg | grep A31 > test.hl7
```

This extracts only A31 messages from the SMAT .msg file.

- ```
msgExtract
bnd01ms4p_in.tin.msg | egrep 'A01|A11' > test.hl7
```

This extracts A01 and A11 messages from the SMAT .msg file.

- ```
msgExtract
bnd01ms4p_in.tin.msg | tr -d '\r'
```

This displays the whole message on the screen, unparsed, without the carriage returns (\r), and with each message on its own line. You can also keep piping to other commands, for example, tail or head.

- ```
msgExtract
bnd01ms4p_in.tin.msg | perl -p -e 's/TEST\^TEST/HELLO\^WORLD/' >
test.hl7
```

This example demonstrates that you can also pipe to perl. Numerous UNIX commands are available to do analysis.

msgReader

This parses a newline-delimited HL7 or a SMAT .msg file into segments. This includes an argument (-w <#>) to set line wrapping.

This splits the messages into segments. As alone this makes messages more readable, when combined with grep or other UNIX tools, message analysis becomes much more powerful.

```
#!/usr/bin/perl -lw
#####
#
# Name: msgReader
# Purpose: Parse HL7 into segments
#
#####

use Getopt::Std;
use Text::Wrap;
```

```
getopt('w');  
  
if ($opt_w) {  
    $Text::Wrap::columns = $opt_w;  
}  
  
$/ = "\r";  
$i = 0;  
  
while (<>) {  
    s/\n//g;  
    if (/^MSH/) {  
        $i++;  
        print "\nMessage: $i\n";  
    }  
    if ($opt_w) {  
        print wrap("", "    ", $_);  
    } else {  
        print;  
    }  
}
```

msgReader examples

msgReader examples include:

- `msgExtract bnd01ms4_in.tin.msg | grep 'TEST\^' | msgReader`

In this example, you first `msgExtract` the messages so that you can `grep` out `TEST` messages, as `grep` operates on newlines.

The results are then piped to `msgReader`.

```

Message: 1
MSH|^~\&|MS40C|100|CERNER|100|20080905060049||ORR^O02|00000000001786872|T|2.3
MSA|AA|
PID|1|000700228|444756||TEST^MCKESSON^^^|19451004|F||W|9000 PHARMACY ROAD^^LAND
LOW^MO^65214^^^|
ORC|NA|00010|7789399237789399||IP|N|1^ONCE^^200809050600^9|200809050600|350987^MAR||00428^PE
TERS^W
OBR||00010|7789399237789399|0337345^POTASSIUM^K^^|
TER^R^^MD|
ZOD|01030649620001100033734523778939923778939900000000101
Message: 2
MSH|^~\&|MS40C|100|CERNER|100|20080905061606||ORR^O02|00000000001786873|T|2.3
MSA|AA|
PID|1|000647254|444635||TEST^TERESA^X^^|19720716|F||W|123 ANY STREET^^SAINT PE
TERS^MO^63376^^^ST CH
ORC|NA|00003|7789284237789284||IP|N|1^ONCE^^200809050715^9|200809050615|350987^MAR||00431^SEE^WILL
OBR||00003|7789284237789284|0383224^TROPONIN T-QUANTITA
TIVE^^TROP^T^^|
ZOD|01030650710001100038322423778928423778928400000000101
Message: 3
MSH|^~\&|MS4ADT|100|CLOVERLEAF|100|20080905085815||ADT^A28|00000000001786887|P|2.3
EVN|A28|20080905085812||BOMS4TEST
PID|1|000740037|445977||HORIZON^TEST^^^|19540504|M||W|SOMEWHERE
STREET^^COLUMBIA^MO^65201^^^BOONE|B
GT1|1|000740037|HORIZON^TEST^^^|
STREET^^COLUMBIA^MO^65201^^^BOONE||19540504|M||A|993

```

- `msgReader bnd01ms4_in.tin.msg | grep PID`

Result:

```

PID|1|000700228|444756||TEST^MCKESSON^^^|19451004|F||W|9000 PHARMACY ROAD^^LAND
LOW^MO^65214^^^|
PID|1|000647254|444635||TEST^TERESA^X^^|19720716|F||W|123 ANY STREET^^SAINT PE
TERS^MO^63376^^^ST CH
PID|1|000740037|445977||HORIZON^TEST^^^|19540504|M||W|SOMEWHERE
STREET^^COLUMBIA^MO^65201^^^BOONE|B
PID|1|000740037|445977||HORIZON^TEST^^^|19540504|M||W|SOMEWHERE
STREET^^COLUMBIA^MO^65201^^^BOONE|B
PID|1|000740037|445977||HORIZON^TEST^^^|19540504|M||W|SOMEWHERE
STREET^^COLUMBIA^MO^65201^^^BOONE|B
PID|1|000740037|445977||HORIZON^TEST^^^|19540504|M||W|SOMEWHERE
STREET^^COLUMBIA^MO^65201^^^BOONE|B
PID|1|000740037|445977||HORIZON^TEST^^^|19540504|M||W|SOMEWHERE
STREET^^COLUMBIA^MO^65201^^^BOONE|B
PID|1|000740037|445977||HORIZON^TEST^^^|19540504|M||W|SOMEWHERE
STREET^^COLUMBIA^MO^65201^^^BOONE|B
PID|1|000700228|444756||TEST^MCKESSON^^^|19451004|F||W|9000 PHARMACY ROAD^^LAND
LOW^MO^65214^^^|

```

- `msgReader bnd01ms4_in.tin.msg | egrep 'Message|ZIP'`

This shows the message # and the ZIP segments. You can use `egrep` to narrow down the search.

```

Message: 1
Message: 2
Message: 3
Message: 4

```



```

Message: 5
ZIP|10.128.18.166|BOMS4TEST|PHRE010|~~~|
Message: 6
ZIP|10.128.18.166|BOMS4TEST|HSRE019HL7|~~~|
Message: 7
ZIP|10.128.18.166|BOMS4TEST|PHRE010|~~~|
Message: 8
ZIP|10.128.18.166|BOMS4TEST|HSRE019HL7|~~~|
Message: 9
msgReader -w 70 bnd01ms4_in.tin.msg

```

- This example demonstrates the `-w` wrapping argument. You can set any number. When setting the wrapping and then piping to `grep`, `grep` does not only show the matching lines, as in this example:

```

Message: 1
MSH|^~\&|MS40C|100|CERNER|100|20080905060049||ORR^002|000000000017868
72|T|2.3
MSA|AA|
PID|1|000700228|444756||TEST^MCKESSON^^^|19451004|F||W|9000 PHARMACY
ROAD^^LANDLOW^MO^65214^^^|||M|UNK|00260049580|158963111
ORC|NA|00010|7789399237789399|IP|N|1^ONCE^^200809050600^^9||20080905
0600|350987^MAR||00428^PETERS^WALTER^R^^^MD|4000
OBR||00010|7789399237789399|0337345^POTASSIUM^K^^^|||^^^|00 428^PETERS^WAL
TER^R^^^MD|||05000000||1^ONCE^^200809050600^^
9|||^^^
ZOD|01030649620001100033734523778939923778939900000000101

Message: 2
MSH|^~\&|MS40C|100|CERNER|100|20080905061606||ORR^002|000000000017868
73|T|2.3
MSA|AA|
PID|1|000647254|444635||TEST^TERESA^X^^|19720716|F||W|123 ANY
STREET^^SAINT PETERS^MO^63376^^^ST CHARLES|ST
CHARLES|||S|NON|00260051230|999999989
ORC|NA|00003|7789284237789284|IP|N|1^ONCE^^200809050715^^9||20080905
0615|350987^MAR||00431^SEE^WILLIAM^M^^^MD|ED
OBR||00003|7789284237789284|0383224^TROPONIN
T-QUANTITATIVE^^TROPT^^^|||^^^|00431^SEE^WILLIAM^M^^^MD|
|||05000000||1^ONCE^^200809050715^^9|||^^^
ZOD|01030650710001100038322423778928423778928400000000101

Message: 3
MSH|^~\&|MS4ADT|100|CLOVERLEAF|100|20080905085815||ADT^A28|0000000000
1786887|P|2.3
EVN|A28|20080905085812||BOMS4TEST
PID|1|000740037|445977||HORIZON^TEST^^^|19540504|M||W|SOMEWHERE
STREET^^COLUMBIA^MO^65201^^^BOONE|BOONE|||M|UNK|993388371
GT1|1|000740037|HORIZON^TEST^^^|^^^|SOMEWHERE
STREET^^COLUMBIA^MO^65201^^^BOONE||19540504|M|A|993388371|||M|||Y||UNK|||W

```

- `msgReader bnd01ms4_in.tin.msg | grep MSH`

Result:

```
MSH|^~\&|MS4OC|100|CERNER|100|20080905060049||ORR^002|00000000001786872|T|2.3
MSH|^~\&|MS4OC|100|CERNER|100|20080905061606||ORR^002|00000000001786873|T|2.3
MSH|^~\&|MS4ADT|100|CLOVERLEAF|100|20080905085815||ADT^A28|00000000001786887|P|2.3
MSH|^~\&|MS4ADT|100|CLOVERLEAF|100|20080905085900||ADT^A31|00000000001786888|P|2.3
MSH|^~\&|MS4ADT|100|CLOVERLEAF|100|20080905085900||ADT^A05|00000000001786889|P|2.3
MSH|^~\&|MS4ADT|100|CLOVERLEAF|100|20080905085900||ADT^A08|00000000001786890|P|2.3
MSH|^~\&|MS4ADT|100|CLOVERLEAF|100|20080905090026||ADT^A01|00000000001786891|P|2.3
MSH|^~\&|MS4ADT|100|CLOVERLEAF|100|20080905090026||ADT^A08|00000000001786892|P|2.3
MSH|^~\&|MS4OC|100|CERNER|100|20080905103736||ORR^002|00000000001786901|T|2.3
```

- `msgParser`

Parses a newline-delimited or a SMAT .msg file into HL7 fields.

```
#!/usr/bin/perl -lw
#####
#
# Name: msgParser
# Purpose: Parse HL7 into fields
#
#####

# Field delimiter is carriage return for segments
$/ = "\r";
$msgnum = 1;

# Iterate over segments
while (<>) {
    tr/\n//d;
    @fields = split '\\|', $_;
    $name = shift @fields;
    exit unless $name;
    $name =~ tr/\x0B//d;
    $flag = 0;
    $n = 0;

    # Iterate over fields
    for (@fields) {
        tr/\x0D//d;
        tr/\x1C//d;
        $n++;
        if ($name eq "MSH")
        {
            if ($flag == 0)
            {
                print "\nMessage $msgnum\n";
                print "MSH.01: |";
                $flag = 1;
                $msgnum++;
            }
            if ($n < 9)
            {
                $msh_n = $n+1;
                print "$name.0$msh_n: $_" if $_;
            } else
            {
                $msh_n = $n+1;
                print "$name.$msh_n: $_" if $_;
            }
            $flag = 1;
        }
        else
        {
            if ($n < 10)
            {
```

```
        print "$name.0$n: $_" if $_;
    } else
    {
        print "$name.$n: $_" if $_;
    }
}
}
```

msgParser examples

This splits HL7 messages into fields.

You can use this script on SMAT .msg files and files that are generated from msgExtract or msgReader. Then, you can pipe data to it from different places.

This script correctly counts MSH fields starting with the first pipe (|) as MSH.01. MSH.02 is usually "^~\&". Most of the time the pipe is the field delimiter, so this is hardcoded. You can alter the script to work on messages where there is a different delimiter.

The script left pads a 1-digit number with a zero, so that grep can be more accurate and simpler to use. This lines up the fields on the screen.

In these examples, the first thing is to gather messages from a SMAT .msg file and put them in test.hl7.

```
msgExtract bnd01ms4_in.tin.0809062315.msg | grep 'TEST\^'
> test.hl7
```

The remainder of the examples operate on this test.hl7 file.

- msgParser test.hl7

```

Message 1
MSH.01: |
MSH.02: ^~\&
MSH.03: MS40C
MSH.04: 100
MSH.05: CERNER
MSH.06: 100
MSH.07: 20080905060049
MSH.09: ORR^O02
MSH.10: 00000000001786872
MSH.11: T
MSH.12: 2.3
MSA.01: AA
PID.01: 1
PID.02: 000700228
PID.03: 444756
PID.05: TEST^MCKESSON^^^
PID.07: 19451004
PID.08: F
PID.10: W
PID.11: 9000 PHARMACY ROAD^^LANDLOW^MO^65214^^^
PID.16: M
PID.17: UNK
PID.18: 00260049580
PID.19: 158963111
ORC.01: NA
ORC.02: 00010
ORC.03: 7789399237789399
ORC.05: IP
ORC.06: N
ORC.07: 1^ONCE^^200809050600^^9
ORC.09: 200809050600
ORC.10: 350987^MAR
ORC.12: 00428^PETERS^WALTER^R^^MD
ORC.13: 4000
OBR.02: 00010
OBR.03: 7789399237789399
OBR.04: 0337345^POTASSIUM^^K^^
OBR.15: ^^^^^
OBR.16: 00428^PETERS^WALTER^R^^MD
OBR.24: 05000000
OBR.27: 1^ONCE^^200809050600^^9
OBR.39: ^^^^^
ZOD.01: 01030649620001100033734523778939923778939900000000101

Message 2
MSH.01: |
MSH.02: ^~\&
MSH.03: MS40C
MSH.04: 100
MSH.05: CERNER
...
```

- msgParser test.hl7 | egrep '^\$|Message|MSH.09|PID.05'

This shows blank lines: Message #, MSH.09, and PID.05.

```

Message 1
MSH.09: ORR^O02
PID.05: TEST^MCKESSON^^^

Message 2
MSH.09: ORR^O02
PID.05: TEST^TERESA^X^^
```

```

Message 3
MSH.09: ADT^A28
PID.05: HORIZON^TEST^^^

Message 4
MSH.09: ADT^A31
PID.05: HORIZON^TEST^^^

Message 5
MSH.09: ADT^A05
PID.05: HORIZON^TEST^^^

Message 6
MSH.09: ADT^A08
PID.05: HORIZON^TEST^^^

Message 7
MSH.09: ADT^A01
PID.05: HORIZON^TEST^^^

Message 8
MSH.09: ADT^A08
PID.05: HORIZON^TEST^^^

Message 9
MSH.09: ORR^O02
PID.05: TEST^MCKESSON^^^

```

- `msgParser test.hl7 | egrep 'PID.05'`

This shows PID.05 only.

```

PID.05: TEST^MCKESSON^^^
PID.05: TEST^TERESA^X^^
PID.05: HORIZON^TEST^^^
PID.05: HORIZON^TEST^^^
PID.05: HORIZON^TEST^^^
PID.05: HORIZON^TEST^^^
PID.05: HORIZON^TEST^^^
PID.05: HORIZON^TEST^^^
PID.05: HORIZON^TEST^^^
PID.05: TEST^MCKESSON^^^

```

- `msgParser test.hl7 | egrep 'MSH.09' | sort -u`

This gets a list of event types, sorted uniquely.

```

MSH.09: ADT^A01
MSH.09: ADT^A05
MSH.09: ADT^A08
MSH.09: ADT^A28
MSH.09: ADT^A31
MSH.09: ORR^O02

```

- `msgParser test.hl7 | egrep '2008' | sort -u`

This searches for date stamps in the message.

```

EVN.02: 20080905085812
EVN.02: 20080905085858
EVN.02: 20080905085859
EVN.02: 20080905090024
MSH.07: 20080905060049
MSH.07: 20080905061606
MSH.07: 20080905085815
MSH.07: 20080905085900
MSH.07: 20080905090026
MSH.07: 20080905103736
OBR.27: 1^ONCE^^200809050600^^9
OBR.27: 1^ONCE^^200809050715^^9
OBR.27: 1^ONCE^^200809051036^^9
ORC.07: 1^ONCE^^200809050600^^9
ORC.07: 1^ONCE^^200809050715^^9
ORC.07: 1^ONCE^^200809051036^^9
ORC.09: 200809050600
ORC.09: 200809050615
ORC.09: 200809051036
PV1.44: 200809050859
PV1.44: 200809150700

```

- `msgParser test.hl7 | egrep 'ORC.01'`

This only shows ORC.01. This is simpler to search for with the left padding. ORC.01 would match on ORC.11, ORC.12, ORC.13, and so on.

```

ORC.01: NA
ORC.01: NA
ORC.01: NA

```

- `msgParser test.hl7 | egrep 'ORC.10'`

```

ORC.10: 350987^MAR
ORC.10: 350987^MAR
ORC.10: 350829^STE

```

- `msgParser test.hl7 | egrep 'ORC.(01|10)|OBR.16'`

This shows ORC.01, ORC.10, and OBR.16.

```

ORC.01: NA
ORC.10: 350987^MAR
OBR.16: 00428^PETERS^WALTER^R^^MD
ORC.01: NA
ORC.10: 350987^MAR
OBR.16: 00431^SEE^WILLIAM^M^^MD
ORC.01: NA
ORC.10: 350829^STE
OBR.16: 00428^PETERS^WALTER^R^^MD

```

Using xlateInVals and xlateOutVals

`xlateInVals` is the list variable Tcl that is used to store the values from the source side of the xlate statement.

`xlateOutVals` is the list variable Tcl that is used to store the values from the destination side of the `xlate` statement.

Note: These variables are list objects and not strings.

This example converts an address to all uppercase letters in an `xlate`.

In Tcl:

```
lappend xlateInVals "1111 Chapel Hill Rd."
set xlateOutVals [string toupper [lindex $xlateInVals 0]]
puts [lindex $xlateOutVals 0] ;# Returns "1111" not "1111 CHAPEL HILL RD."
set tmp [string toupper [lindex $xlateInVals 0]]
set xlateOutVals [list $tmp]
puts [lindex $xlateOutVals 0] ;# Returns "1111 CHAPEL HILL RD."
```

The first `puts` statements returns only "1111". The second `puts` statement returns the entire address because `xlateOutVals` is treated similar to a list.

This is added to the Translation Configurator's Action pane:

```
BULKCOPY
COPY: {0(0).PID(0).#11(0).(0) -> 0(0).PID(0).#11(0).(0)}
COPY: {0(0).PID(0).#19(0).(0) -> 0(0).PID(0).#19(0).(0)}
```

In this example, `xlateOutVals` is not treated as a list in the Pre Proc pane.

```
set tmp [string toupper [index $xlateInVals 0]]
```

The result is 1111 in PID.11.

```
{0(0).PID(0).#11(0).(0) : >1111^^COLUMBIA^M0^65203
```

You can also treat `xlateOutVals` as a list.

```
set tmp [string toupper [index $xlateInVals 0]]
set xlateOutVals [list $tmp]
```

The result is 1111 CHAPEL HILL RD. in PID.11.

```
{0(0).PID(0).#11(0).(0) : >1111 CHAPEL HILL RD.^^COLUMBIA^M0^65203
```

Passing multiple values in and from Tcl fragments

Because `xlateInVals` and `xlateOutVals` are lists, you can pass multiple values in and from Tcl fragments.

1 In the Action pane, specify:

```
BULKCOPY
COPY: (Hello, World!) (0(0).PID(0).#11(0).[0]) -> @someVar (0(0).PID(0).#11(0)[0])
COPY: @someVar {~0(0).PID(0).#11(0).[0]} -> ?
COPY: {0(0).PID(0).#19(0).(0) -> 0(0).PID(0).#19(0).(0)}
```

- 2 With the first `COPY` statement highlighted (Hello, World! . . .), add to the Pre Proc pane:

```
set tmp1 [index $xlateInVals 0]
set tmp2 [string toupper [index $xlateInVals 1]]
set xlateOutVals [list $tmp1 $tmp2]
```

- 3 With the second `COPY` statement highlighted (@someVar . . .), add to the Pre Proc pane:

```
puts "someVar: [lindex $xlateInVals 0]"
puts "PID.11: [lindex $xlateInVals 1]"
```

- 4 The final result is:

```
someVar: Hello, World!
PID.11: 1111 CHAPEL HILL RD.
```

Accessing Tcl help

Tcl help is available from the tcl prompt by specifying `help`. The help is especially useful for `hci` commands.

If you cannot remember the entire command, then entering part of the command at the prompt retrieves usage help.

Fetching value

Note: “Field” and “sub-field” are used for HRL/VRL/FRL formats. For HMD formats, replace “field” with “component” and “sub-field” with “subcomponent”.

The retrieval (fetching) value that is based on a field-level address has changed from CIS 5.8 to CIS 6.0 and later versions.

A change has been made in how the engine retrieves values from a field-level address as the source side in a translation action.

When the sub-field-level address is used as a source of a `COPY` action, the engine now retrieves the whole sub-field content. This includes all sub-sub-fields as a string. By doing this, it does not copy only the first sub-sub-field.

The entire content as a string is assigned to `xlateInVals`.

The current behavior is where the engine retrieves a value from a field-level address as the source side in a translation action.

Before CIS6.0, when setting a field-level address as the source in a translation action and the field had sub-fields, the engine would expand the field. Then, the engine would retrieve all the sub-field's content, and put the values in a list.

This led to these issues:

- If there was only one destination, then only the first sub-field content was copied to that destination.
- If there are several destinations, then the sub-field's content was copied to incorrect destinations.

This issue was fixed in CIS 6.0. When the field-level address is set as a source, the engine retrieves the whole field content. This includes all its sub-fields and sub-field separator characters as a string. The entire field content can be copied to the destination.

Fetching examples

In this example, VRL has a **Test** field that contains three sub-fields. The sub-field separator character is “-”.

Field "Test2" does not have any sub-fields.

The input content is:

```
Test: >A-B-C<
Test2: >D<
```

One source and one destination

```
{ { OP COPY }
  { ERR 0 }
  { IN Test }
  { OUT @tmp }
}
```

Before CIS 6.0, @tmp was . Only the first sub-field was copied.>A<

After CIS 6.0, @tmp is >A-B-C<. The entire field content is copied.

Two sources and two destinations

```
{ { OP COPY }
  { ERR 0 }
  { IN {Test Test2} }
  { OUT {@tmp @tmp2} }
}
```

Before CIS 6.0:

- @tmp was >>A<.
- Only the first sub-field was copied.
- The @tmp was >B<.
- The second sub-field was copied to this destination, which was the content of field Test2(D).

After CIS 6.0:

- @tmp is >A-B-C<.
- The entire field content is copied. The @tmp2 is >D<.
- The content of field Test2 content is copied.

xlateInVals

The content of xlateInVals in Pre Proc is updated.

```
{ { OP COPY }
  { ERR 0 }
  { PRE {
    #Note: Shows $xlateInVals
    puts "xlateInVals is: $xlateInVals"
    lassign $xlateInVals a b c
    puts "$a:$b:$c"
  }}
  { IN Test }
  { OUT @tmp_test }
}
```

Before CIS 6.0:

- xlateInVals was list A B C.
- Using lassign assigned the sub-field's content to variables a,b, and c.

This displayed:

```
xlateInVals is: A B C
A:B:C
```

After CIS 6.0:

- xlateInVals is a string A-B-C.
- Using lassign assigns A-B-C only to variable a, and makes b and c empty strings.

This now displays:

```
xlateInVals is: A-B-C
A-B-C::
```

This change happens on HRL/VRL/FRL and all HMD formats.

Updating an older translation file to work under the new fetch behavior

To copy only the sub-field content, use the sub-field-level address as the source instead of the field-level address.

For example:

```
{ { OP COPY }
  { ERR 0 }
```

```

    { PRE {
#Note: Shows $xlateInVals
    puts "xlateInVals is: $xlateInVals"
    }}
    { IN Test.[1] }
{ OUT @tmp_test }
}

```

This copies only the second sub-field content B to the destination; `xlateInVals` is B only.

To copy the whole field content, but have written a pre-proc to handle the `xlateInVals` list, keep using the field-level address in the source. Then, update the pre-proc.

An example of a convenient method to update your proc is:

```

{ { OP COPY }
  { ERR 0 }
  { PRE {
#Note: Shows $xlateInVals

    puts "The original xlateInVals is: $xlateInVals"
    regsub -all {-} $xlateInVals { } xlateInVals      <-- Replace
the sub-field separator char with space, so that xlateInVals will become a list.
    puts "The xlateInVals after regsub is: $xlateInVals"
    lassign $xlateInVals a b c
    puts "$a:$b:$c"
  }}
  { IN Test }
  { OUT @tmp_test }
}

```

This shows:

- The original `xlateInVals` is A-B-C
- The `xlateInVals` after `regsub` is A B C
- A:B:C

Using Tcl string map

The `string` commands have been optimized for performance.

For finding and replacing characters or strings, `string map` is the most efficient method.

Unless you must find and replace a string based on a pattern, using `string map` is more efficient than `regsub`.

```

set var "000-111-2222"
regsub -all -- {\-} $var "" var
puts $var ;# Returns "0001112222"
set var "000-111-2222"
set var [string map {- {} } $var]
puts $var ;# Returns "0001112222"

```

The `string map` command is simpler to use with special characters and escapes. In the example above, the dash has to be escaped in the `regsub`, but not in the `string map`.

This is an example using `string map` with multiple pairs:

```
set map [list \
  | \\F
  ^ \\S
  \\ \\E
  & \\T
  ~ \\R
  \r \\X0D\\X0A\\
  \n \\X0A\\
]
set var [string map $map $var]
```

External message flags bit masks

This table lists the externally visible bit values that are visible through Tcl.

Bit	Tcl Flag Name	Tcl Writable
0x00000001	should_be_freed	FALSE
Used by internal memory management.		
0x00000002	icl_owns_data	FALSE
Whether the local copy of the message can be destroyed if sent to a thread in another process.		
0x00000004	expect_reply	TRUE
Automatically set when the thread is configured to await replies. When set, the threads await a reply after writing the message through the protocol connection.		
0x00000008	is_forwarded	FALSE
Set if the message is ever forwarded from one thread to another.		
0x00000010	is_enqueued	FALSE
TRUE when MSG is on a queue; otherwise, FALSE.		
0x00000020	last_in_group	TRUE
Set to indicate that this message is the last in a related group.		
0x00000040	proto_timeout	TRUE
Set by the protocol driver to describe a protocol-send failure due to timeout. Intended for use by SENDFAIL TPS procedures.		
0x00000080	proto_nak	TRUE
Set by the protocol driver to describe a protocol-send failure due to nak. Intended for use by SENDFAIL TPS procedures.		
0x00000200	is_resent	TRUE
Set if this message was resent back into the engine.		
0x00000800	recovered	TRUE
Recovers the message from the database.		
0x00001000		

```

is_on_disk
FALSE
0x00002000
keep_on_disk
FALSE
0x00008000
use_rdb
FALSE
Set to indicate that this message will go to the recovery database if there is any engine
failure.
0x00010000
prewrite_done
FALSE
Set if this message has a prewrite procedure.

```

Upgrading encodings to MB or 6.2 and later versions

This only applies to situations where a user-configured encoding transformation (using `hcitpstblconvert` or `msgmapdata`) is used in an earlier Cloverleaf version. That is, any encoding performed in Tcl using `hcitpstblconvert`, `msgmapdata` or any other method, including the standard ones provided. This does not apply for any other use.

Users with SNA interfaces communicating to EBCDIC (Extended Binary Coded Decimal Interchange Code) systems must be aware of this.

Two options are available for upgrading non-standard encodings to multi-byte or CIS:

- Option 1: Use the existing `hcitpstblconvert`.
Select the binary encoding option from the Network Configurator.
If the IB message data is not a standard encoding supported by the IBM ICU library and is being translated by a user Tcl proc, then the IB encoding binary should be selected for the IB protocol thread. By selecting binary encoding, you can continue to use the user-configured encoding transformation table of the earlier version.
Note: The binary option must be used if the ICU library does not support encodings found in the current and previous Cloverleaf versions of Tcl. IBM037 is a common standard encoding.
- Option 2: Remove the existing `hcitpstblconvert`.
Add the correct encoding to the `.ini` file so that it displays on the Network Configurator's **Encoding** menu. Then, you can configure the thread to use the newly added encoding.

For example, IBM037 is not on the encoding list, so it must be added. To do this:

- 1 Navigate to `server/i18n.ini`.
- 2 Add **IBM037** after ASCII. It is then available on the **Encoding** menu.

Configuring a standard inbound TCP/IP server thread

- 1 On the Network Configurator's **Properties** tab, select the **pdl-tcpip** protocol.
- 2 On the **PDL TCP/IP Protocol Properties** dialog box, configure:

- a For **PDL** select `mlp_tcp.pdl`.
 - b For **Type** select **Server**.
 - c Specify a port.
- 3 On the **Network Configurator Inbound** tab, for **TPS Inbound Data** click **Edit** and select `hl7Raw_ack` from the **Inbound TPS Editor** dialog box.

Configuring a standard outbound TCP/IP client thread

- 1 On the Network Configurator's **Properties** tab, select the `pdl-tcpip` protocol.
- 2 Click **Properties** to open the protocol's dialog box:
 - a For **PDL** select `mlp_tcp.pdl`.
 - b For **Type** select **Client**.
 - c Specify a host and port.
- 3 On the **Inbound** tab, select the **Outbound Only** check box.
- 4 On the **Outbound** tab:
 - a Select **Await Replies**.
 - b For **Timeout**, specify **45**, the default.
 - c For **TPS Inbound Reply**, select `hcitpsmsgkill`.
 - d For **Trx ID Determination Format** select **HL7**.
- 5 Click **Timeout Handling** and click **Resend OB message**.

Using the Fileset protocols without a custom TPS procedure

Without any procs at dirparse or delete, the Fileset and Fileset FTP protocols contain every file that is in the references directory. Every retrieved file is deleted after it is processed.

When this is not the case, you must have Tcl procs at the directory parse or file delete UPoCs. For example:

- You only require a few files in the directory at directory parse.
- You do not require files that are retrieved during directory parse, then processed deleted.
- You do not require to rename, move, or archive the files before processing, after directory parse or before deletion.

Example:

There are no procs. Directory "A" has files "a," "b," and "c" at the time the directory parse is run in the Fileset protocol.

In this case, each of the files is read, and each record in each file becomes a message to the system.

As a file is finished processing (file "a" first), that file is then deleted. The records of that file have passed through the system to where they were directed. File "a" is processed and deleted. File "b" is processed and deleted. File "c" is processed and deleted.

Directory parse happens at the next interval and if there are more files they are processed in the same manner.

Deleting a single message

You can delete a single message from the recovery or error database using the message ID:

```
hcidbdump -r -m xxxxxxxx -D
```

xxxxxxx is the message ID. This is the number you see after the 0.0. in `hcidbdump -r`.

To determine which message to delete, use the `UltraLong` display to look at the contents of the messages. When you have identified the message, you can use the above command to delete it.

You also must stop the associated thread to clear the message, as the engine may have a copy of the message in memory.

Using UltraLong format

To look at the message ID in `UltraLong` format in the background without using the dialog box, use:

```
hcidbdump -r -L
```

The message ID is on a line that says `msgMid`.

Performing search functions

You can also perform search functions in the database by putting the output in a file for searching:

```
hcidbdump -r -L > error.out
```

Then you can use a viewer to look and search for entries in that file. For example, `vi`, `cat`, `pg`, and so on. Or, you can FTP the file to another server and use a text editor such as Notepad or Wordpad.

Use `-d` to add the destination or `-f` to add the source to keep the file specific for what you are searching.

Recovery database lock

To avoid locking the recovery database as this command runs, which causes messages to be un-deliverable, add the `-u` switch to the command:

```
hcidbdump -r -U username -L > recoverydb.txt
```

The username that you use is not that important but keeps it from using the default user name that the engine uses.

Setting up UNIX users

Use these steps to set up UNIX users to start, stop, and run system commands from the shell command line without privilege issues.

When setting up UNIX users, remember these points:

- The `sudo` and `su` commands provide a comprehensive audit trail.
 - The command that is issued along with the issuer's user name is logged to the file `/var/log/secure`.
 - Each successful authentication is logged to the file `/var/log/messages`.
- 1 Use `visudo` to add the `staff` group to the `sudo` list by adding `%staff ALL=(ALL) ALL`.
 - 2 The `"s"` sets the `setgid` flag on the directory so that new files inherit the group ownership. For example:

```
find $HCIR00T -type d -exec chmod g+s {} \;
```

- 3 Add all Cloverleaf users and `hci` to the `staff` group. For example:

```
usermod -a -G staff user
```

- 4 Disable direct SSH log-in for the `hci` user. To do this:
 - a Add the line `DenyUsers hci` to `/etc/ssh/sshd_config`.
 - b Restart `sshd` using `service sshd restart`.
- 5 Make `hciengineerun`, `hcienginestop`, `hcisitectl`, and `hciss` readable, writable, and can run only to `hci`. For example:

```
chmod 700 $HCIR00T/bin/hciengineerun
$HCIR00T/bin/hcienginestop chmod 700 $HCIR00T/bin/hciengineerestart
$HCIR00T/bin/hcisitectl $HCIR00T/clgui/bin/hciss
```

You can now only start and stop Cloverleaf processes as `hci`.

Stopping and starting Cloverleaf processes should always be performed by the `hci` user.

- 6 Because the `hci` user log-in is disabled, files must be transferred as the user. To ease permission issues, use a transfer directory.

- a As hci:

```
mkdir $HCIR00T/site_master/data/transfer
```

- b As hci:

```
chmod 777 $HCIR00T/site_master/data/transfer
```

- c As user: Transfer files to and from this directory.
d As hci: Use the `cp` command to copy the files into place.
e As hci: Use the `rm` command to delete the files from the transfer directory.

Adding/removing hciss auto-starting on UNIX/Linux

You can use `hciunixservice` to add/remove auto-starting of `hciss` on UNIX/Linux systems.

See [hciunixservice](#) on page 1394.

Installing the host server as a service in UNIX/Linux

- 1 Log in to root using `su` or `sudo`.
- 2 Change the directory, using `cd $HCIR00T/bin`.
- 3 Run `hciunixservice -i -h` to install the host server as a service.
Replace `-i` with `-r` to remove the service.
Replace `h` with `s` to install or remove the security server as a service.
- 4 Run `hostserver service start` to start the service.
When shutting down the system, the host/security server and processes, if assigned, gracefully shut down and auto-start after the system boots.

Recovery database

The system engine uses the recovery database to manage information about messages that the engine is currently processing. These messages can be automatically recovered if the system crashes. As a message enters the system, it is assigned a unique message ID and copied to the recovery database.

Then, after each stage of processing, a new version of the message replaces the existing one. In this way, the recovery database always has the most current version. After a message has been completely processed by the engine, it is removed from the recovery database.

Operator guide

Cloverleaf Integration Services (CIS) facilitates the exchange of information among the various applications within an organization. CIS enables all these systems to communicate with each other, even though they use different information formats and different communications protocols.

Operators monitor these tasks through limited system tools usage.

This section provides Operators the information required for the day-to-day operation and maintenance of CIS in the supported environments.

Threads and processes

A thread is a separate container that holds one running copy of the engine. Any number of threads can be active simultaneously. The number of active threads is only limited by the amount of available RAM.

A group of threads that work together make up a process:

- One command thread manages the overall operation of the process. It controls all other threads within a process.
- One translation thread controls message translation. There is only one translation thread per process. This thread transforms messages from the format of the sending system to the format of the receiving system.
- The protocol thread manages protocol connections to remote systems. One or more protocol threads receive messages from an inbound host connection and pass them to the translation thread. They can also receive messages from the translation thread and pass them to an outbound host connection. The same protocol thread can perform both operations.
- A connection is the interface between an external system and a protocol thread. Because there is a thread associated with every connection, the two terms are sometimes used interchangeably.

The only threads that you can directly control are the protocol threads. You can start and stop individual protocol threads, and all the threads that make up a process. Threads can be defined for a whole site, or in a predefined or ad hoc group.

Components

These components are contained in the system:

- The host server is the primary working component. It holds all the program files and configuration information required to perform message brokering functions.
- The client is the user interface to the host server.
- The security server is an optional service that provides advanced security features.

The system must have at least one host server and one client. If there is only one client, then it must run on the same machine as the host server.

A typical system is composed of one host server and several clients. One client runs on the same machine as the host server and the others run on connected machines.

The system may or may not have a security server. If there is a security server, then it must be connected to the host server, but it usually runs on a separate machine. A client is never directly connected to the security server.

Tools

Operator control over system tasks is provided through the system tools. You access these tools through the Integrated Development Environment (IDE) or the command line.

- To open the IDE from the command line, specify **hciaaccess** or from Windows, select **Start > All Programs > Infor Cloverleaf Integration Suite > Cloverleaf IDE**.
- From the Launch Bar, under **Runtime** select the tool to open.

You can also select tools from the **Tools** menu.

Each tool operates independently of all the others, so you can simultaneously open multiple tools on the IDE.

When you right-click a tool tab, a menu opens with these options:

- **Close file name**
- **Close Others**
- **Close All**
- **Save file name**

This performs the selected operation on multiple files (tabs) without having to right-click each individual tab.

Cloverleaf system

The system's main purpose is to facilitate the exchange of information among the various applications within your organization. The system enables these applications to communicate with each other, even though they use different information formats and different communications protocols.

The system acts as a hub for exchanging messages, which are the records created by one or more remote systems. In a typical exchange, messages are created in one format in a remote system and sent through the system. This reformats the messages and transmits them through a different protocol to another remote system.

For example, when a patient is admitted to a hospital, an admission record is created in HL7 format. This is sent to the system through an LU 6.2 connection. Information from that same record might later be required in the hospital lab. This lab uses an FRL format and the Fileset FTP protocol. To enable the lab to use the contents of the record, a bridge is provided in the system between the two protocols. This translates between one record format and the other.

The system engine completes several functions. It receives inbound messages and acknowledges receipt. These are ACKs and NAKs. It then extracts the message contents from inbound protocol envelopes, and routes messages to and from the translation system.

The engine also inserts the contents of the translated messages into outbound protocol envelopes and transmits outbound messages to their destinations. The engine maintains message copies in a recovery database until message receipt is acknowledged by the destination applications. It also maintains an error database of messages that have some form of error, and a log of every system event.

Network Configurator

The Network Configurator is used to configure connections to and from the system. This tool, and other configuration tools, are accessed through the IDE.

The IDE provides the rules that enable the system engine to handle both inbound and outbound protocols. It also specifies which of the available translations are to be performed by the translation system.

Selecting the environment

Users have the option of selecting another environment from which to work.

- 1 Select **Server > Change** on the IDE menu bar.
A message opens to verify the server change.
- 2 Click **Yes**.
This opens the **Select a Server and Environment** dialog box. The bottom pane lists the available environments on the host machine.
- 3 Select an environment and click **Apply**.

Client users can access any listed environment immediately without any halt in production. It is not necessary to stop and restart any tools or the host server.

Site Daemons

Site daemons enable you to control two background processes that run on the host server:

- The monitor daemon is essential to the operation of the Network Monitor. The Network Monitor depends on the monitor daemon to send instructions to the system engine and to receive information from the engine.

Therefore, the monitor daemon must be started before you start the Network Monitor.

- In a UNIX system, the Lock Manager controls the locking and unlocking of files. This prevents files from being accessed by more than one user at a time, thus safeguarding the files from corruption.

In a UNIX system, therefore, the Lock Manager must also be started before you start the Network Monitor.

Note: In a Windows system, files are automatically locked and unlocked by the operating system, so there is no requirement for a Lock Manager.

Use the Site Daemons tool to:

- Start and stop one daemon or both daemons together. See [Starting the Site Daemons tool](#) and [Stopping the site daemons](#).
- View the status of daemon operations. See [View the Site Daemon status](#).
- View a log file that contains a record of engine activities. See [View the log file of engine events](#).
- Select an alert configuration file or remove the current one. See [Select an alert configuration](#).

Accessing the site daemons

In a UNIX system, you must start both site daemons before starting the Network Monitor. This is because the Monitor Daemon must be running for the Network Monitor to communicate with the engine. The Lock Manager must also be running to protect any files opened by the Network Monitor.

In a Windows system, only the Monitor Daemon must be started before starting the Network Monitor. This is because it is the only available site daemon.

Start one or both daemons from the Site Daemons GUI, which you can access in any of the standard ways.

- 1 Open the IDE, using one of these methods:
 - From the command line, enter `hciaccess`.
or
 - From Windows, select **Start > All Programs > Infor Cloverleaf Integration Suite > Cloverleaf IDE**.
- 2 From the Launch Bar, select **Site Daemons** from the Runtime list.

Starting the Site Daemons tool

Start one or both daemons from the Site Daemons GUI, which you can access in any of the standard ways.

See [Accessing the site daemons](#).

- 1 After you have accessed the Site Daemons, start them with this procedure, depending on your operating system:
 - (UNIX only) In the **Site Daemons** dialog box, select the daemons to start.
 - In Windows, **Monitor Daemon** is the only available daemon, and is automatically selected.
- 2 Click **Start Daemons**.
After you have started the Monitor Daemon, it continues to run until you stop it or until you shut down the system engine.

Network Monitor controls

The controls for starting and stopping the Lock Manager and Monitor Daemon are conveniently located on the Network Monitor toolbar.

Note: The Lock Manager is available only for UNIX hosts and is not visible when connected to a Windows host.

Start or stop the daemons by clicking the daemon and choosing **Start** or **Stop** on the menu.

If the MonitorD is not running, then the option is **Start**.

If the MonitorD is running, then the options are **Restart** and **Stop**.

Stopping the site daemons

Stopping the Monitor Daemon does not stop the system engine or the Network Monitor. It breaks the communications link between them. With the Monitor Daemon stopped, the Network Monitor cannot send instructions to or receive information from the engine.

Caution: Never stop the Lock Manager when system processes are running.

- 1 Stopping the site daemons depends on your operating system.
 - (UNIX only) In the **Site Daemons** dialog box, select the daemons to stop.
 - In Windows, **Monitor Daemon** is automatically selected.
- 2 Click **Kill Daemon**.

Parameter pane

This table shows the options in the parameter pane:

Option	Description
All Daemons	Click to select all daemons (UNIX).
Monitor Daemon	Click to select the Monitor Daemon only.
Lock Manager	Click to select the Lock Manager only (UNIX).
Alert Cfg	Click to open the Please select an Alert Configuration dialog box, where you can select the alert file. The file name displays in the field to the right.
Clear Cfg	Click to remove the current alert selection from the field.

Command pane

This table shows the options in the command pane:

Option	Description
Start Daemon	Click to begin the action.
Kill Daemon	Click to stop the action.
Restart Daemon	Click to restart the daemons.
Status	Click to show the results of each action. Results display in the Status pane.
View Logfile	Click to open the log file selected from the list.

Site daemon operations

Along with starting and stopping the Monitor Daemon and the Lock Manager, there are other tasks that you can perform from the Site Daemons tool.

These tasks are the same no matter which daemons are selected.

Selecting an alert configuration

An alert is a notification that the Monitor Daemon sends whenever some predetermined condition is met.

An alert configuration is a file that identifies a specific set of alerts.

For example, it might show a warning message when the number of messages in the error database exceeds a specified threshold.

Alerts are the responsibility of the administrator of your system. Any messages that you receive should contain clear instructions for what to do or who to contact.

Your system might have different alert configurations for different sites, different processes, different times of day, and so on. You also might require to switch from one alert configuration to another.

As with alerts, your system administrator is responsible for providing you with instructions for selecting alert configuration files.

Note: Always select the appropriate alert configuration file before you start the Monitor Daemon. The first time you open the **Site Daemons** dialog box during a session, no alert configuration file is selected.

Use this procedure to select an alert configuration file at this time and to change to a different one at any time:

- 1 Stop the Monitor Daemon if it is running.
- 2 Click **Alert Cfg**. This opens the **Please select an Alert Configuration** dialog box.
- 3 In the list of alert configurations, click the one to select.
- 4 Click **Apply**. This returns you to the **Site Daemons** dialog box, with the selected configuration file shown in the text box beneath the daemon selection option buttons.

To clear the current alert configuration, click **Clear Cfg**. This empties the text box.

Viewing the Site Daemon status

Whenever you start or stop a site daemon, your instructions and their results are reported in the Status pane. If the daemon starts or stops without your intervention, then the event is not reported.

Click **Status** to ensure that the status report is complete and current.

Viewing the log file of engine events

The engine maintains a log file that records engine output during the current session.

Click **View Logfile** to open the **Logfile** dialog box.

The text between the brackets provides identifying information about the entry. For example, `[cmd:cmd:INFO/0:hcimonitor <date>/<time>]` has this information:

- The module and submodule that generated the entry, such as `cmd:cmd`.
- The severity level and engine output level of the event, such as `INFO/0`.
- The thread and, if applicable, the message ID, such as `hcimonitor`.

Viewing the MonitorD error log

A menu is available for log selection for the alert log, monitorD error log, and monitorD log. Make a selection and click **View Logfile**.

By default, the standard log is selected, which is the log file opened after clicking **View Logfile**.

Site daemons from the command line

To run or modify Site Daemons from the command line, open a shell window and use the `hcisitectl` command.

```
hcisitectl [-f] [-h host] [{ -K | -k daemon }] [-v]
[{ -S | -s daemon}] [{ -R | -r daemon} -d delay interval]
[-u #users] [-A args] [-n]
```

- `-f` forces stopping the daemons.
- `-h host` is the name of the remote host.
- `-K` stops all daemon processes.
- `-k daemon` stops the specified daemon.. For *daemon*, use a comma-separated list if specifying multiple daemons, where:
 - `l` is the Lock Manager.
 - `m` is the Monitor Daemon.
- `-v` checks for duplicate `hcimonitor`d when starting/stopping MonitorD, preventing duplicate MonitorDs in one site.
- `-s` starts all daemon processes.
- `-s daemon` starts the specified daemon. For *daemon*, use a comma-separated list if specifying multiple daemons, where:
 - `l` is the Lock Manager.
 - `m` is the Monitor Daemon.
- `-R` restarts all daemon processes and is used with `-d delay interval`.
- `-r daemon` restarts the specified daemon and is used with `-d delay interval`. For *daemon*, use a comma-separated list if specifying multiple daemons, where:
 - `l` is the Lock Manager.
 - `m` is the Monitor Daemon.
- `-d delay interval` is the time interval for engine restarting.
- `-u #users` is the maximum number of users for Lock Manager. *#users* defaults to 500 and must be greater than 200.
- `-A args` specifies the startup arguments for the daemons. For *args*, use a comma-separated list if specifying multiple arguments. Each entry has the form *daemon = args*, where:
 - `l args` is the Lock Manager, followed by arguments.
 - `m args` is the Monitor Daemon, followed by arguments.
 - *args* are the arguments to pass to that daemon.
- `-n` specifies to not run the engine in a service on Windows.

If no options are specified, then the status of the daemons is verified and reported.

Note: Do not stop the Lock Manager without first ensuring that there are no running engine processes in the site.

Order of running

As the errdb alert type requires database access, during start-up the Lock Manager should be started before starting the Monitor Daemon.

For shutdown, the Monitor Daemon should be shut down first, followed by the Lock Manager.

Starting all daemons

Use the `-s` argument to start all the daemons.

Starting a specific daemon

To start a specific daemon with an alert, use this syntax. In this example, the alert file is named `foo.alert`.

```
hcisitectl -s m -A "m=-cl foo.alert"
```

For MonitorD, the available parameters are the same as `hcimonitorD`:

- `-de eoc-pattern` enables an engine output configuration pattern.
- `-dd eoc-pattern` disables an engine output configuration pattern.
- `-cl cfg-list` specifies a list of alert configuration files to process. Use a space to separate multiple file names in the list. If no file name is specified, then `default.alert` is processed, if it exists.
- `-nl` enables non-daemon mode. No log file is created. In this case, the Monitor Daemon runs in the foreground. This option is only used for debugging.
- `-D` enables debugging information.
- `-S sitename` is the site name.

Starting a specific alert

To run a specific alert, use this syntax. In this example, the alert file is named `foo.alert`.

```
hcisitectl -s a -A "a=-cl foo.alert"
```

Network Monitor

The Network Monitor enables you to control engine operations by managing and manipulating the connections handled by protocol threads within processes.

Note: A maximum of one (1) Network Configurator can run on the IDE at any given time.

You can use the Network Monitor to:

- Start and stop processes and threads.
- Show status information about processes and threads.
- Hold and release outbound messages.
- Show graphics exhibiting the activity of processes and threads.
- Forward messages to a different thread.
- View event scheduling information.

Starting the host server

Use these steps to start the host server.

Note: In basic or no security and running without an audit server, the IDE can be opened in local mode without a running host server. In advanced security, the IDE cannot run in local mode without a running host server.

- 1 Change to the root directory.
- 2 Specify `hciss -s h`.

Starting and stopping the Lock Manager and Monitor Daemon

The controls for starting and stopping the Lock Manager and Monitor Daemon are conveniently located on the Network Monitor toolbar.

To perform its tasks, the Network Monitor uses the Monitor Daemon, which passes information and instructions between the Network Monitor and the engine.

Therefore, the Monitor Daemon must be started before you start the Network Monitor.

In a UNIX system, the Network Monitor depends on the Lock Manager. This prevents files from being accessed by more than one user at a time, safeguarding them from corruption. In UNIX systems, the Lock Manager must be started before you start the Network Monitor.

The controls for starting and stopping the Lock Manager and Monitor Daemon are located on the Network Monitor toolbar.

Start or stop the daemons by clicking the daemon indicator and choosing **Start** or **Stop** on the menu.

Note: The Lock Manager is available only for UNIX hosts and is not visible when connected to a Windows host.

- If the corresponding daemon is alive, then the indicator for that daemon is green.
- If the daemon shuts down or Network Monitor loses its connection, then the indicator is red.

If the MonitorD/Lock Manager is running, then a **Restart** option is available. When restarting the MonitorD/Lock Manager, the GUI cannot be operated until the restart has completed. This is because the MonitorD/Lock Manager stops and then restarts.

An error message opens if the restart is unsuccessful.

Delay interval

Restarting Delay is a customizable delay interval for the waiting period between stopping and starting when restarting MonitorD, Lock Manager, a process, or a thread.

This is found on the **General** tab of the **Root Preferences** dialog box. This is accessed at **Options > Root Preferences**.

MonitorD error status and log

When an error condition exists and the connection between the Network Monitor GUI and MonitorD is on, an error status displays on the Status bar.

The IDE gets the MonitorD status from a response from the `iwant` command, which is provided by the engine.

Pausing the mouse on the error icon opens a tool tip that shows how many errors exist, along with a Click to view message. Clicking the message opens the **Alert Error Detail** dialog box, which shows detailed error information.

The error icon is not dismissed until the Network Monitor tool is closed or the MonitorD is not on error status anymore.

The MonitorD standard log is viewed from the Site Daemons GUI.

On the Site Daemons GUI, there is a menu from which you can select the standard log or error log.

When a Monitor Daemon option is selected, options are:

- **Alert Log**
- **MonitorD Error Log**
- **MonitorD Log**

By default, the standard log is selected, which is the log file opened after clicking **View Logfile**.

Starting Network Monitor

The Network Monitor cannot access the engine unless the host server is running. See [Starting the host server](#).

Note: Before starting the Network Monitor, you must start the appropriate site daemons.

To access the Network Monitor:

- 1 Open the IDE, using one of these methods:
 - From the command line, enter `hciaccess`.
or
 - From Windows, select **Start > All Programs > Infor Cloverleaf Integration Suite > Cloverleaf IDE**.
- 2 From the Launch Bar, select **Runtime** and click **Network Monitor**.

Network Monitor tabs

The **View** tab is the default tab when the Network Monitor first opens. It shows a visual map of selected threads and provides complete control over them. Tab colors apply to the individual route lines.

The depth color is automatically applied to the route lines on the tab.

For example, the queue depth is two each of connections and routes. This is four in the total per-thread queue. These can be a different color.

The **List** tab shows a table that lists all the threads in the current view, with information about each thread.

The entry for each thread includes:

- Thread name.
- Thread status.
- Volume. This is the number of messages processed by the thread during the current session.
- Depth. This is the number of messages in the thread's message queue.

The **Grid** tab shows the thread name and status, protocol status, depth number and color bars that indicate the protocol status (optional) and depth.

The current depth color is based on a summary of the thread depth and route depth. The thread depth number is the depth of the thread. The route depth number is the summary of the route depth from the target thread.

In a cell, the tip box shows the detail depth information as follows:

- Thread: thread_name
- Thread Depth: thread_depth_number
- Route Depth: route_depth_number

The thread depth and route depth are displayed separately.

Each cell shows the thread status icon, thread name, protocol status, and depth number. The color of the line under the depth number represents the depth color setting.

Thread names that are too long are truncated.

The more threads there are, the more columns there are, with a maximum of eight columns. In these cases, the thread name could be truncated.

Double-clicking in a grid cell opens the **Status Report** dialog box for that thread.

View and Options menus

This table shows the **View** menu options:

Option	Description
Command and Engine Output	<p>Opens the Command and Engine Output panel. In this panel, you can view the log file of engine output during the current session. You can also view command operation output, and errors and warnings that are defined on the Engine Output tab of the Network Monitor Properties dialog box.</p> <p>See Command and engine output.</p>
Process Control	<p>Opens the Process Monitor dialog box. By default, the Process Monitor opens along the right side of the Network Monitor. Use this dialog box to view process indicators for all processes included in the current view. These indicators provide direct control over the operation of the processes.</p> <p>See Process monitor.</p>
Alerts	<p>Opens the Alerts dialog box. This lists all the alert messages that have been sent but not yet deleted. The listing for each alert includes the date sent; source host, root, and site; and the complete text of the alert message.</p> <p>See Alerts.</p>
Create View	<p>Creates a new view. Select the threads to include in the view to enable this option. Then click Create View to open the Create View dialog box. You can also right-click a selected thread to access this option.</p>
View List Pane Show All Reply Routes Show All Routes	Switches the selected option.
Filter View	Opens the Thread Filter Settings dialog box.

Options menu

On this menu, selecting **Network Monitor Options** opens the **Network Monitor Properties** dialog box.

Use this dialog box to monitor and configure polling intervals, alert settings, engine output filters, and depth colors.

Command and engine output

Click **View > Command and Engine Output** to open the Command and Engine Output panel.

In the Command and Engine Output panel, you can view command and engine output for each process in the network configuration.

This panel contains several notebook tabs, depending upon the number of processes.

The **Command Output** tab shows the output of any command operation. Click **Clear** to remove the currently shown output.

Process monitor

By default, the Process Monitor panel opens along the right side of the Network Monitor.

The Process Monitor panel lists the processes in the current view.

These indicators provide direct control over the operation of the processes.

Processes are sorted by process name in ascending order.

This panel is vertically resizable. If the name is too long for the pane width, then a horizontal scroll bar displays.

To open a context menu of process management commands, right-click a process indicator. This opens the context menu for process management.

Process control context menu

Right-click a process indicator to open the process context menu. This table shows the options on the context menu:

Option	Description
Control	This opens the process control context menu, where you can: <ul style="list-style-type: none">• Start, Stop, Restart the selected process.• Select Full to open the Process Controls dialog box.• Select All Processes to open a menu where you can Start, Stop, or Restart all processes at one time.
Metrics	Opens the Engine Monitor dialog box, where you can review performance statistics for the session. You can also view graphs that show the performance in specific areas over time.
Conn Status	Opens the Status Report dialog box for the selected process.
Status	Opens the Process Status Summary dialog box.
Watch Output	Shows process output as it happens.

Option	Description
Browse Output	Opens the log file of process output.
Startup Log	Opens the Startup Log for Process dialog box.
Notes	Notes can be viewed in plain text or HTML. Notes are written on the Network Configurator's Process Configuration dialog box.

Starting and stopping multiple processes

Right-clicking in the Process Control pane, but not on any process, opens a menu with options for starting, restarting, or shutting down multiple processes.

Selecting **Start**, **Stop** or **Restart** opens the **Select Processes to Start/Stop/Restart** dialog box. This dialog box is a list of processes. The title depends on the menu selection.

When you select **Restart**, this dialog box lists only the running processes.

Multiple processes can be selected from the list. After you click **OK**, the IDE starts/stops/restarts the selected processes. The order to start/stop/restart the processes is the same as the order of the selected processes on the list. The command line output is appended to the Command and Engine Output pane, where you can see the results.

You can also right-click any of the processes to access the **Control > All Processes** menu, which opens the same **Select Processes to Start/Stop/Restart** dialog box.

- If no process is running, then only **Start** is available.
- If all processes are running, then **Stop** and **Restart** are available.
- If some processes are running and some are dead, then **Start**, **Stop**, and **Restart** are all available.

Alerts

Click **View > Alerts** to open the **Alerts** dialog box. This dialog box shows all triggered alert messages that have been generated but not yet deleted. Selecting an alert from the list shows the details.

This table lists the fields on the **Alerts** dialog box:

Field	Description
First Received	Shows the date and time the alert was first received. This may be the same as Last Received .
Last Received	Shows the date and time the alert was most recently received.

Field	Description
Duplicates	Shows the number of duplicates of the alert that have been received. If no duplicates have happened, then this value is 0.
Host	Lists the host where the alert was generated.
Root	Lists the root for which the alert was generated.
Site	Lists the site for which the alert was generated.
Type	The alert type. For example, thread status .
Name	The alert name.
Repeat Count	Shows the repeat configuration when the check box is selected.
Alert field	Shows the text of the alert message.
Alert list	This is a summary of all the alerts that have been received since Network Monitor was started. By default, the alert list is sorted by date last received. Alerts which have yet to be reviewed or which have a newly-arrived duplicate display in bold.
Remove Selected	Removes the currently selected alert from the alert list. This is available only when at least one alert is selected.
Save Selected	Saves the selected alert to a text file located on the local machine. This is available only when at least one alert is selected.
Close	Closes the dialog box until opened by the user or until a new alert causes it to open automatically.
Status bar	Indicates the current total number of alerts and the number of unread alerts, if any.

Alert list

The bottom pane of the **Alerts** dialog box shows all triggered alert messages. Initially, when the dialog box is first opened, the top message is automatically selected. A different message or multiple messages can also be selected.

Use the buttons along the bottom of the dialog box to delete or save the selections to a text file.

If a message listing is in bold, then it indicates one of these:

- That message has not yet been read.
- A duplicate of that message has arrived.

Alert list order

By default, the list is sorted in chronological order.

Click **Shift+column heading** to change the default listing order.

Click the column heading again to return to the default order.

If multiple messages are selected, then the top pane keeps the information from the previously selected single message.

Alert notification

When Network Monitor receives notification of a new alert from the Monitor Daemon, the **Alerts** dialog box opens and the new alert is selected.

You can change this behavior by selecting the appropriate check box on the **General** tab of the **Network Monitor Properties** dialog box.

To do this, select **Options > Network Monitor Options** to open the **Network Monitor Properties** dialog box.

This table shows the available options:

Alert notification type	Description
Automatically Display New Alerts	<p>Select this check box to automatically show new alerts through the Alerts dialog box. This is the default.</p> <p>Clear the check box to disable this feature. If this check box is cleared, then Automatically Select New Alerts is also cleared. In this way, there is always some notification of a new alert, through the Alerts dialog box or the Network Monitor status bar icon.</p>
Automatically Select New Alerts	<p>Select this check box to automatically select new alerts so that the details are visible. This is the default.</p> <p>Clear the check box to disable this feature. If this check box is cleared, then when a new alert arrives an icon displays on the Network Monitor status bar. This indicates that there is an unread alert.</p> <p>Click the icon to open the Alerts dialog box. If this dialog box is already showing, then it is brought to the front. The alert icon remains on the status bar until all alerts have been read.</p>

Note: If the Alerts dialog box is minimized, then Network Monitor cannot restore it when a new alert arrives.

Network Monitor Properties dialog box

Select **Options > Network Monitor Options** to open the **Network Monitor Properties** dialog box.

Configuration options are found on these tabs:

- **General**
- **Engine Output**
- **Depth Colors**
- **Protocol Colors**
- **Grid View**

General tab

The polling intervals determine how often Network Monitor checks threads and site daemons for status.

This table lists the available options:

Option	Description
Statistics Updates	Specifies the thread polling interval. Click the arrow to select from the list.
Monitor Daemon Polling	Specifies the Monitor Daemon polling interval. Click the arrow to select from the list.
Lock Manager Polling	Specifies the Lock Manager polling interval. This option is available only for UNIX hosts.
Automatically Display New Alerts	<p>Select this to automatically show new alerts. This is the default.</p> <p>Clear the check box to disable this feature. When this check box is cleared, Automatically Select New Alerts is also cleared. When this is cleared, users always have notification that a new alert has arrived. Notification comes through the Alerts dialog box or the Network Monitor status bar icon.</p>

Option	Description
Automatically Select New Alerts	<p>Select this to automatically select new alerts so that the details are visible. This is the default.</p> <p>Clear the check box to disable this feature. When this check box is cleared and a new alert arrives, an icon displays on the Network Monitor status bar. This indicates that there is an unread alert.</p> <p>Click the icon to open the Alerts dialog box. When this dialog box is open, it is brought to the front. The alert icon remains on the status bar until all alerts have been read.</p> <p>If the Alerts dialog box is minimized, then Network Monitor cannot restore it when a new alert arrives.</p>
Show notes as HTML by default	<p>Notes can be written in plain text or HTML and can also be viewed on the Network Monitor's Process Configuration dialog box. In this case, the Network Monitor's Notes tab shows the notes for the selected process.</p> <p>To view the Thread or Process notes in the Network Monitor, select the Notes option on the thread icon's right-click menu.</p> <p>When you select Notes, a dialog box opens to show the notes which are added and saved in Network Configurator.</p> <p>These notes are read-only. Click Done to close the dialog box. The Notes dialog box supports some HTML features, including hyperlink.</p>
Only load the last 10 lines of a process log	<p>To keep from having to resize the Watch Log File for process dialog box (Control > Full), you can select this option. When this is selected, the dialog box loads only the last 10 lines and remembers the last opened size.</p> <p>The default is cleared. These functionalities also apply to the Message Archive Log Viewer.</p>

Engine Output tab: Adding a filter

Use this tab to specify which messages are shown or ignored.

To add a new engine output filter:

- 1 In the **Regular Expression** text box, specify the regular expression or string contained in the output message to filter.
- 2 Select the action for filtering the output (**Display** or **Ignore**).

- 3 (UNIX) To choose different colors for displayed output, specify the name or hexadecimal value of any valid UNIX color in the **Foreground** text box.
- 4 Specify a comment in the **Comment** text box describing the filter.
- 5 Select the **Number of lines displayed per process** using the up or down arrows.
- 6 Select the **EO Update Interval** in seconds using the up or down arrows.
- 7 Click **OK** to add the new filter.

Depth Colors tab

Use this tab to determine and select various message queue depth colors for the **Grid View** tab. These are the number of messages queued for the connection, but not yet delivered.

View tab colors apply to the individual route lines.

Grid tab colors apply to the thread.

The depth color is automatically applied to the route lines on the **Grid View** tab.

The Depth pane lists the various queue depths to which colors have been assigned.

This table lists the depth options:

Option	Description
Add Depth	Adds another depth to the listings on the Depth pane. For example, you could add a listing for "equal to or less than 6."
Delete Depth	Deletes the highlighted listing in the Depth pane.

Click a queue depth color to edit its color in the **Choose Thread Color** dialog box. This is where you can select a color swatch or specify the particular color.

The color that is shown is the current color for the selected queue depth. **Reset** restores to the default settings.

Protocol Colors tab

The protocol status color is shown in a rectangular bar under the protocol status message and is colored according to the protocol status.

This protocol status color is updated when any protocol status is changed.

By default, every protocol color is distinct from the default depth colors of red, green, and yellow.

All protocols have the same status color settings.

The protocol status color is set on the **Protocol Colors** tab of the **Network Monitor Properties** dialog box.

The protocol status color settings are user-specified client settings, and are saved in `client.ini`.

For example:

```
[user_<name>_100.0.0.0_6.2_chapard_netmonitor]
protocolcolors=true
protocolcolors_file=-8355712 -16777216 -16777216 -65536 -16731648 -10040065 -10040065 -16777038
-20561
protocolcolors_tcpip=-16737895 -16777216 -16777216 -16777216 -16731648 -10040065 -10040065 -
16777038 -20561
```

The value for the `protocolcolors` key indicates the **Display protocol status color** status:

- `true` means display.
- `false` means do not display.

After `protocolcolors` are the protocol status color settings for the individual protocols.

- The protocol name is in the key. This must be lowercase.
- The protocol status color settings for individual protocols following the fixed prefix `protocolcolors_`.
- The color setting is a number that represents the RGB color value in the default RGB integer. Bits 24-31 are alpha, 16-23 are red, 8-15 are green, and 0-7 are blue.
- The delimiter among colors are an empty space.
- The color values are in a fixed sequence for individual protocol status.
- If a protocol is not found here and `protocolcolors` is true, then the default color is used.
- If no `protocolcolors` setting is found, then no protocol status color is shown. In this case, it works the same as having `protocolcolors` be false.

Setting the protocol status color

- 1 Select **Display Protocol Status Color** to configure the protocol color. This shows the protocol color bar on the thread icon. This check box is not selected by default, indicating no protocol status is shown on the thread icon.
- 2 Select the protocol from the **Protocol** list. This lists all possible protocols.
- 3 The table in the remainder of the dialog box shows the selected colors for individual protocol statuses. This table has two columns, Protocol Status and Color.
Click the column header to sort the columns.
- 4 Click a color's button to open the **Color Chooser** dialog box. In this dialog box, you can select the color for the protocol status on that same row. The color cell shows the selected color as its background color. You can set a different color for each individual protocol status.
- 5 Click **OK** to commit the new settings. These settings are immediately applied to the thread icons in the View tab.

Grid View tab

This table shows the options for configuring the displayed thread name and protocol status on the **Grid** tab:

Option	Description
Show thread name	Shows the thread name on the Grid tab.

Option	Description
Show protocol status	Shows the protocol status on the Grid tab.
Preview	Shows a preview of how the grid display settings are shown on the Grid tab.

Selecting thread control in a process

To open a dialog box that has full control of all the threads in a process:

- 1 Right-click a process name on the Process Monitor panel. This opens the context menu for process management.
- 2 Select **Control** to open the process control context menu.
- 3 Click **Full**. This opens the **Process Controls** dialog box for the selected process.

Process Controls dialog box options

Right-click a process to open the context menu. Selecting **Control > Full** to open the **Process Controls** dialog box.

This table lists the dialog box options:

Option	Description
start	Initiates the process by starting all the threads in the selected process.
stop	Shuts down the process by stopping all the threads in the selected process.
start with hold	Starts the threads in the process, with the outbound data on hold.
start with sticky hold	Starts the threads in the process with the last hold status.
restart	Restarts the process. When restarting a process, the GUI can still be operated. The command line output is appended to the Command and Engine Output pane, where you can see the results. When restarting a process, the thread status icon on the GUI changes from green to red to green again.
hold	Places a hold on all outbound data queues for all protocol threads in the process.

Option	Description
release	Releases held messages, so they can continue processing for all protocol threads in the process.
hold reply	Holds transmission of all thread reply messages of the selected process until released.
release reply	Releases hold on all or a specified number of reply messages.
reload	Reloads updated Tcl procedures.
purge caches	Removes the configuration caches. Use this option after making a change to an existing configuration. This is not designed to be used for the creation of new threads, processes, routes, and so on.
watch output	Shows engine output as it happens.
browse output	Opens the log file of engine output.
engine log configuration	Opens the Engine Log Configuration dialog box. Click List to select the engine output alias. This applies an Engine Output (EO) alias, configured in the Engine Output Configurator, to the threads in the process.
reindex	Loads Tcl procedures that are created after the engine began running.
cycle output	Causes the current engine log (.log) and error (.err) files to close and be renamed with an .old extension, and then opens new logs.
status	Opens the Status Report dialog box, which provides statistical information on the threads within the process.
reset statistics	Resets statistics and counts for a site's process or thread. This calls the <code>hcimsiutil</code> engine command. Before using this option, there is the potential to produce unreliable counts. The memory is not corrupted when this option is used. The engine and Network Monitor both lock the shared memory for any write/clear.

Managing views and threads

The Network Monitor offers many ways to manage threads. A straightforward approach is to select a view in the Network Monitor. Then, you can use the context menu on the View List pane to manage all the threads in the view together. You can also use the context menu for a thread icon on the **View** tab to manage an individual thread.

The View List pane lists all the views that have been created for the current site. Each view is a visual map of a selected set of threads. Threads that share a part of the same route are shown linked to each other.

The first, default, view in the list is "all_threads". This view shows all the threads in the current site. It is created automatically and cannot be removed.

Views are also created automatically for every process and every group that has been configured in the Network Configurator. These views are named *process_name* or *group_name*.

Thread icons

Each thread is represented by a thread icon.

The thread indicator shows the current status:

- Green indicates the thread is running.
- Red indicates the thread is not running.
- A blue outline around the icon indicates the focus.
- On the destination site, if there is any route to a public thread, an icon is shown on the upper-right of the public thread. This indicates the referenced public thread. The referenced indicator shows only after synchronization is completed from the source site.

The thread connection status is any one of:

- Dead: The thread is not running. It has stopped or been shut down.
- Initializing: The thread has been started and the engine is preparing to run it.
- Opening: The thread is attempting to establish a connection.
- Up: The thread is connected and running.
- Timer: An event is scheduled for that thread.
- Down: The thread is running, but is not attempting to establish a connection.
- Closing: The thread is in the process of shutting down the connection.
- Error: The thread connection has failed because of a problem.
- INEOF: A file-based connection has reached the end of the input file.

The thread name uniquely identifies the thread.

The protocol status color is shown in a rectangular bar under the protocol status message and is colored according to the protocol status.

- The protocol status color is updated when any protocol status is changed.
- By default, every protocol status color is distinct from the default depth colors of red, green, and yellow. All protocols have the same status color settings.

- The protocol status color is set on the **Protocol Colors** tab of the **Network Monitor Properties** dialog box.

Route lines

If two or more of the threads share a part of the same route, then the threads are shown linked by an arrow.

The arrow shows the direction of message flow from thread to thread.

The arrow color represents queue depth. The colors can also be customized to your site through the **Choose Thread Color** dialog box. This dialog box is accessed by clicking **Change** on the **Depth Colors** tab of the **Network Monitor Properties** dialog box.

Route lines stay visible whether the route line is clicked. To find out more about a route, right-click the route line, and then click **Transaction Summary** to open the **Transaction Summary** dialog box. This dialog box shows:

- The number of messages sent across the route during the current session.
- The number of messages waiting in queues.
- The name of the translation thread.
- The type of translation.
- The file that contains the translation specifications.

Click **Refresh** to refresh the run-time route summary.

Reply route representation

Route lines are depicted by a solid line.

Reply route lines are dashed.

Depth colors are also applied to reply routes. For example, the color of any reply route is updated based on the reply queue depth.

View pane

A view pane is located on the left side of the Network Monitor. This pane displays as the view pane in the Network Configurator. The view pane is a horizontally resizable pane in Network Configurator and Network Monitor.

The view pane lists all views in the current `mvw` file in sorted order. These views can only be single-selected. The selected view is shown in the layout pane. By default, the "all_threads" view is automatically selected.

If any view is changed without being saved, then an asterisks (*) is appended to the view file name in the title.

Types of views

There are different types of views in the view pane:

- Built-in views include the "all_threads" view, process views, and group views. The "all_threads" view contains all threads in the Network Configurator. The IDE creates a process view for each process. The process view name is named with the syntax `process_process_name`.
- You can view threads/processes/groups in user-created views. These can contain a sub-view which references other views.

If a thread is created in a group or process view in Network Configurator, then the group/process of the thread is what is currently selected.

The IDE also creates a group view for every group and is named with the syntax `group_group_name`. All default created views are refreshed based on the Network Configurator file `group_group_name` when Network Configurator/Network Monitor is opened.

Network Configurator can be changed separately from the view file. The views in the earlier view file might not be consistent with the later Network Configurator file, especially for default created views. The refresh is performed to make sure the default created views are up-to-date.

For the "all_threads" view and user-created views in the Network Configurator, the default process name of a new thread is the site name. The default group name is blank.

You can change the process/group of a thread in a selected view in Network Configurator. If the new process/group does not belong to the selected view, then there is a warning message after you click **OK** to commit the change:

The new process/group of the thread `thread_name` does not belong to current view. The `thread_name` will be removed from this view.

After closing the warning message, the changed thread is removed from the current view and is added to the view to which it belongs.

Starting or stopping all threads in a view

- 1 Select the view that contains the threads to start, stop, or restart. To do this, click the view name in the View List pane and select "all_threads".
- 2 Right-click anywhere in the View List pane to open the context menu for view management.
- 3 Select **Control** to open the view control context menu.
- 4 Click **Start** to start all threads, **Stop** to stop all threads, or **Restart** to restart all threads.

Starting or stopping an individual thread

- 1 Select the view that contains the thread to start, stop, or restart by clicking the view name in the View List pane.
- 2 Select the thread to start, stop, or restart. To do this, right-click the appropriate thread icon on the **View** tab. This opens the context menu for thread management.

- 3 Select **Control** to open the thread control context menu.
- 4 Click **Start** to start the thread, **Stop** to stop the thread, or **Restart** to restart the thread.

Context menu for view management

To open a context menu of view management commands, select a view and right-click anywhere on the View List pane. This opens the context menu for view management.

This table shows the options on the context menu:

Option	Description
Control	Opens the view control context menu. You can Start , Stop , or Restart the threads that are shown in the view. You can also select Full to open a dialog box of options.
Metrics	Opens the Engine Monitor dialog box. You can re-view performance statistics for the session, and view graphs that show the performance in specific areas over time.
Status	Opens the Status Report dialog box.
Edit View	Opens the Create View dialog box with the specifications for the selected view. Use this dialog box to add or delete threads or other views.
Delete View	Deletes the selected view, after prompting you to confirm your intention.
Filter View	Opens the Thread Filter Settings dialog box. Use this dialog box to control which threads display in the current view.
Add New View	Opens the Create View dialog box with no view specifications. In this dialog box, you can create a new view that contains the selected threads or views.
Import View	Opens a file browser, from which you can select a view to import. The filter settings in the <code>.view</code> file are based on the current <code>NetConfig</code> file during importing. All threads in the view file and in the current <code>NetConfig</code> file are imported into a new single view configuration.

Using the view control context menu

Use the view control context menu to access view-specific commands for the selected view.

- 1 On the Network Monitor, click a view in the view list.
- 2 Right-click anywhere on the View List pane to open the context menu for view management.
- 3 Select **Control** to open the view control context menu with these options:
 - **Start** the threads included in the selected view.
 - **Stop** the threads included in the selected view.
 - **Restart** the threads included in the selected view. When restarting a view, the GUI can still be operated. The command line output is appended to the Command and Engine Output pane, where you can see the results.
 - **Full** opens the **View Controls** dialog box.

Creating a new view

- 1 Right-click in the View List pane. This opens the context menu for view management.
- 2 Click **Add New View**. This opens the **Create View** dialog box.
- 3 Specify a view name in the **View Name** text box.
- 4 Select the threads or views to be included in the new view.

Editing an existing view

- 1 Right-click an existing view on the View List pane. This opens the context menu for view management.
- 2 Click **Edit View**. This opens the **Create View** dialog box for the selected view.
- 3 Edit as necessary.

Updating views

Use Network Configurator to create new threads or to update existing properties.

When a thread is updated or added to an existing process, that thread is automatically added to the corresponding process view in Network Configurator.

For group views, threads that are updated or added to an existing group are also added to the group view in Network Configurator.

Note: Built-in views, including the all_threads view, process view, and group view, cannot be updated or edited.

In Network Monitor, you must refresh the view to see the change (**File > Refresh**).

Accessing the View Controls dialog box

To open a dialog box that offers full control of all the threads in a view:

- 1 In the Network Monitor, click a view in the view list.

- 2 Right-click anywhere in the View List pane to open the context menu for view management.
- 3 Select **Control** to open the view control context menu.
- 4 Click **Full**. This opens the **View Controls** dialog box.

View Controls dialog box options

This table lists the options on the dialog box:

Option	Description
start	Starts all the threads in the selected view.
stop	Stops all the threads in the selected view.
start with hold	Starts the threads in the view, with the outbound data on hold.
start with sticky hold	Starts the threads in the view with the last hold status.
restart	Restarts all the threads in the selected view.
hold	Holds outbound messages in the outbound queue.
release	Releases any messages currently being held.
hold reply	Holds transmission of all thread reply messages for the selected view until released.
release reply	Releases hold on all or a specified number of reply messages.
reload	Adds any new Tcl procedures.
purge caches	Removes the configuration caches. This action clears all record and lookup table configurations. Use this option after making a change to an existing configuration.
engine log configuration	Enables the modification of engine output for the current run. This applies an Engine Output (EO) alias, configured in the Engine Output Configurator, to the threads in the view.
reindex	Loads Tcl procedures that are created after the engine begins running.
status	Opens the Status Report dialog box.

Option	Description
reset statistics	Resets statistics and counts for a site's process or thread. This calls the <code>hcimsiutil</code> engine command. Before using this option, there is the potential to produce unreliable counts. The memory is not corrupted when this option is used. The engine and Network Monitor both lock the shared memory for any write/clear.

Thread management options

To open a context menu of thread management commands, right-click anywhere on a thread icon.

This table lists the options on the context menu for thread management.

Option	Description
Control	Opens the Thread Control Context menu. You can Start , Stop , or Restart the selected thread. You can also select an action, or select Full to open a dialog box of options.
Metrics	Opens the Engine Monitor dialog box. You can review performance statistics for the session, and view graphs that show the performance in specific areas over time.
Status	Opens the Status Report dialog box.
SMAT	<p>Open Inbound and Open Outbound open a SMAT Database tab to the <code>*_in.smatdb</code> or <code>*_out.smatdb</code> file.</p> <p>The file type opened depends on which SMAT driver is being used. The SMAT driver can be changed by selecting/clearing Save into Database on the SMAT tab of the Site Preferences dialog box.</p> <p>If it is selected, then the SMAT database file is opened if it has been configured. Otherwise, the SMAT plain file is opened.</p>
Schedule	<p>This is enabled when advanced scheduling is configured for the thread.</p> <ul style="list-style-type: none"> View opens the Schedule thread dialog box. Run Now runs the schedule.
Notes	Opens the Notes dialog box for that thread. This option is available only when there are notes, indicated by the icon on the thread.

Option	Description
Create View	Opens the Create View dialog box, where you name a new view. Ctrl-click as many threads as are required for that view.

Viewing scheduled events

Events are scheduled for Fileset Local, Fileset FTP, HTTP Client, or UPoC.

- Use the Network Configurator to schedule events.
- Use the Network Monitor to enable or disable events.

When a schedule is set, the timer symbol displays on the thread icon.

Right-click the thread icon to open the context menu with **View Schedule** enabled.

- Select **View Schedule** to view that thread's schedule.
- Use the **Schedule** dialog box to disable or enable that thread's schedule.

The clock icon also displays on the Grid pane.

Accessing the Thread Control context menu

Use the Thread Control context menu to access thread-specific commands for the selected thread.

- 1 In the Network Monitor, click a view in the view list.
- 2 Right-click a thread icon on the **View** tab to open the Context Menu for Thread Management.
- 3 Select **Control** to open the Thread Control Context menu.

Thread Control context menu options

This table lists the options on the Thread Control context menu:

Option	Description
Start	Starts the selected thread. If necessary, then the process containing the thread is started.
Stop	Stops the selected thread. The process containing the thread is not stopped.
Restart	Restarts the selected thread. If necessary, then the process containing the thread is restarted.
Hold	Holds outbound messages in the outbound queue.

Option	Description
Hold Reply	Holds transmission of all reply messages for the selected thread until released.
Release	Releases any messages currently being held in the selected thread.
Release Reply	Releases hold on all or a specified number of reply messages.
Full	Opens the Thread Controls dialog box.

Accessing the Thread Controls dialog box

To open a dialog box that offers full control of a single thread:

- 1 In the Network Monitor, click a view in the view list.
- 2 Right-click a thread icon on the **View** tab to open the Context Menu for Thread Management.
- 3 Select **Control** to open the Thread Control Context menu.
- 4 Click **Full**. This opens the **Thread Controls** dialog box for the selected thread.

Thread Controls dialog box options

This table lists the options on the **Thread Controls** dialog box:

Option	Description
start	Starts the selected thread.
stop	Stops the selected thread.
start with hold	Starts the thread, with the outbound data on hold.
start with sticky hold	Starts the thread with the last hold status.
restart	Restarts the selected thread. When restarting a thread, the GUI can still be operated. The command line output is appended to the Command and Engine Output pane. The thread status icon on the GUI changes from green to red to green again.
hold	Holds outbound messages in the outbound queue.
release	Releases any messages currently being held.
hold reply	Holds transmission of all reply messages for the selected thread until released.

Option	Description
release reply	Releases hold on all or a specified number of reply messages.
start save	Starts the saved message file.
stop save	Stops the saved message file.
cycle save	Causes the current saved message (SMAT) file to close and be renamed with an <code>.old</code> extension, and then opens a new saved message file. Each thread can have at most two SMAT files, one for inbound data and one for outbound data. The names and locations of these SMAT files are user-definable. Note: The engine automatically turns off saved messaging when disk space reaches a certain threshold (90%). It turns on again when the percentage used drops below the threshold. This has no effect on the actual messages successfully sent to the receiving system. Saved messages are only copies of messages sent when saved messaging is active.
resend	Opens the Resend Datafile dialog box. Use this dialog box to insert a set of messages from a stored file into a running engine.
reload	Adds any new Tcl procedures.
purge caches	Removes the configuration caches. Use this option after making a change to an existing configuration.
start forward	Starts forwarding data messages to a configured destination.
stop forward	Stops forwarding data messages.
start reply forward	Starts forwarding reply messages to a configured destination.
stop reply forward	Stops forwarding reply messages.
watch output	Shows engine output as it happens.
browse output	Opens the log file of engine output.
engine log configuration	Enables the modification of engine output for the current run. This applies an Engine Output (EO) alias, configured in the Engine Output Configurator, to the thread.
reindex	Loads Tcl procedures that are created after the engine began running.
status	Opens the Status Report dialog box.

Option	Description
message archive log	Opens the Message Archive Log Viewer for the archive message logs.
open inbound SMAT	<p>Opens the corresponding *_in.smatdb file configured in the Network Configurator's Properties tab, In-bound pane.</p> <p>The file type opened depends on which SMAT driver is being used. The SMAT driver can be changed by selecting/clearing Save into Database on the SMAT tab of the Site Preferences dialog box.</p> <p>If it is selected, then the SMAT database file is opened if it has been configured. Otherwise, the SMAT plain file is opened.</p>
open outbound SMAT	<p>Opens the corresponding *_out.smatdb file configured in the Network Configurator's Properties tab, Outbound pane.</p> <p>The file type opened depends on which SMAT driver is being used. The SMAT driver can be changed by selecting/clearing Save into Database on the SMAT tab of the Site Preferences dialog box.</p> <p>If it is selected, then the SMAT database file is opened if it has been configured. Otherwise, the SMAT plain file is opened.</p>
reset statistics	<p>Resets statistics and counts for a site's process or thread.</p> <p>This calls the hcimsiutil engine command.</p> <p>Note: Before using this option, there is the potential to produce unreliable counts. The memory does not corrupt when this option is used. The engine and Network Monitor both lock the shared memory for any write/clear.</p>

Tracking and reviewing performance

Network Monitor offers many ways to track and review performance:

- Track commands as they are issued to the engine, along with the engine's responses and engine output for each process, by selecting **View > Command and Engine Output**.
- Track and review the operations of one or more threads by opening the **Engine Monitor** dialog box.
- View a current status report by opening a **Status Report** dialog box.
- View alert messages by opening an **Alerts** dialog box.

Reviewing thread metrics

The Network Monitor context menus have a **Metrics** option. This opens the **Engine Monitor** dialog box that contains statistical information about the performance of:

- A selected thread.
- A selected group of threads.
- All the threads in a selected process.
- All the threads in a selected view.

When the **Engine Monitor** dialog box is first opened, it shows statistics that are related to the whole time that the threads have been running.

Engine Monitor menu options: Graphs menu

Right-click a thread to open the context menu. Selecting **Metrics** opens the **Engine Monitor** dialog box. Graphs can be shown individually or with other graphs.

This table lists the graph options:

Option	Description
Outbound Processing Latency	<p>Opens a graph showing the outbound processing latency. Latency refers to the time taken to process a message.</p> <p>This graph has these characteristics:</p> <ul style="list-style-type: none">• The metric unit is in seconds.• The yellow graph is the total latency for messages in the process. This time includes inbound processing, translation, outbound processing, and queue time.• The units on the left side of the graph are an aggregate of ALL messages currently in the engine. If there are 100 messages in outbound queues waiting to be sent, then the number is the sum of all their wait times.
Msgs/Sec	Opens a graph showing the average number of messages processed per second.
Bytes/Sec	Opens a graph showing the average number of bytes processed per second.
Txns/Sec	Opens a graph showing the average number of transactions processed per second.

Engine Monitor menu options: Interval menu

Right-click a thread to open the context menu. Selecting **Metrics** opens the **Engine Monitor** dialog box. Graphs can be shown individually or with other graphs.

The commands in the **Interval** menu set the interval between updates to the information in the **Engine Monitor** dialog box. The available intervals range from 5 seconds (default) to 5 minutes.

Totals pane

This table lists the metrics for the time period:

Named	Description
Txn Count	The total number of processed transactions.
Txns/Sec	The average number of processed transactions per second.
Send Time	The average number of seconds to send a message to its destination, an external connection or translation thread.
Xlt Time	The average number of seconds to translate a message.
Msgs In	The total number of inbound messages that are received.
Bytes In	The total number of bytes that are received for inbound messages.
Msgs Out	The total number of outbound messages that are sent or delivered.
Bytes Out	The total number of bytes that are sent or delivered for outbound messages.
Up	The number of threads that are controlled by this process that are currently running. This statistic is available only when viewing the metrics of a process.
Down	The number of threads that are controlled by this process that are currently not running. This statistic is available only when viewing the metrics of a process.
State	The current status (up, down, and so on) of the selected thread. This statistic is available only when viewing the metrics of a thread.

These metrics are updated at regular intervals. The interval between updates is determined by the selection on the **Interval** menu.

Performance metrics graphics

Each command in the Graphs menu of the **Engine Monitor** dialog box opens a graph of a different metric. Although each graph measures a different aspect, all have the same basic design.

The numbers on the left are ranges for the metrics. They are automatically adjusted to show different ranges for the actual numbers. For example, the graph for bytes per second might range from 4000 to 7000, with a horizontal line at every 500.

The numbers at the bottom are times at which the graph is updated, in hh:mm:ss format. These times depend on the interval selected for updates. The breaks differ slightly because of the small amount of time it takes to process the information and change the display.

Scroll bar

Every time the graph is updated, the newest interval displays at the right side and the oldest scrolls to the left side. To freeze the graph, use the scroll bar at the bottom by scrolling to the left. The graph stays wherever you stop the scroll bar.

Status review

Right-click a thread to open the context menu. Selecting **status** opens the **Status Report** dialog box. Graphs can be shown individually or more at a time.

The status report covers all the threads listed in the title bar of the dialog box, listing a separate report for each thread. By default, the **Status Report** dialog box opens to show the last report in the list.

To update the status report automatically, click **Auto-Update**.

This table lists the status report information for each thread:

Name	Description
Connection	The name of the thread, or connection, for the current status.
Reported at	The date and time of the status report.
Status	The thread status at the reported time.
Proto info	Shows the number of connected clients for the TCP/IP and PDL TCP/IP protocols.

Name	Description
Started/Stopped	The date and time that the thread was last started or stopped. This depends on the status.
Msg in	The number of messages read by this thread from its protocol connection.
Last Rd	The date and time that the messages were last read by this thread from its protocol connection.
Bytes in	The number of bytes read in by this thread from its protocol connection.
Msgs out	The number of messages that are written out by this thread to its protocol connection.
Last wt	The date and time that the messages were last written out by this thread to its protocol connection
Bytes out	The number of bytes that are written out by this thread to its protocol connection.
Proto Info	The number of connected clients for the TCP/IP and PDL TCP/IP protocols.
Proto Err	The date and time of the last protocol error.
Error Msg	The text message of the last protocol error.
Xlated	The number of messages that are translated by this thread.
Pending	The number of pending messages for this thread. For example, messages waiting to be delivered to a destination thread or outbound connection that is down.
Forwarded	The number of messages forwarded to an alternate host. This is after the failure of all transmission re-tries to the primary destination host.
Failed	The number of messages that were not delivered successfully.
IB Lat	Inbound Latency. The number of cumulative seconds that this thread has taken to process inbound messages.
OB Lat	Outbound Latency. The number of cumulative seconds that this thread has taken to process and deliver outbound messages.
Total Lat	The total of inbound and outbound latency in seconds This is cumulative.

This table shows the statistics for each thread interaction with the other threads that are selected for this report:

Name	Description
Sent	The number of messages that are sent by this thread.
Recvd	The number of messages that are received by this thread.
pxqd	Pending Xlate Queue Depth. The number of messages, or queue depth, that the thread has sent to its destination translation thread that have not yet been processed.
Xlt time	The average number of seconds required to translate a message.
T on Q	Time on Queue. The total number of seconds spent waiting on queues. This is T on Q. This is updated when the message is delivered. T on Q values exist only for threads which have delivered messages. Compare this value to "Total Lat" to evaluate the amount of processing time versus the pending time.
Latency	The total number of seconds that the thread spent on the same last message, from receipt to delivery. Subtract "T on Q" from "Tot Lat" to find the processing time.
Thread name	The name of the sending or receiving thread.

Resolving issues

This section is intended for analysis, information gathering, and basic troubleshooting for operators using the system. Use the information in this section to diagnose and resolve problems before they become emergencies.

If you are in a production down emergency, then the information provided by these tools can be useful to Support.

Contacting Support

In most cases, you should report any problems that you encounter in day-to-day system operations to the administrator of your organization's system. To obtain technical advice and information from Support, select **Help > Support** on the IDE for contact information.

Running low on disk space

If you are running low on disk space, then you can find out which files are using more disk space and recover.

To prevent the recurrence of this issue, recycle the `.log` and SMAT files on a routine basis, as part of preventative maintenance. This applies to all system supported platforms.

Reclaiming disk space in UNIX

- 1 Go to the `/cisversion/integrator/` directory and run the `df -k` command. This command lists the size and percentage of space for each file system on your machine.
- 2 Run the command `du -sk *`. This gives a list of all of the directories and files, with their current size in kilobytes.
- 3 Change to the directory that is too large. This is usually `/integrator`. Then, run the command `du -sk *` again. Repeat this until you locate the files that are too large. These are usually `.log` files in the process directory, or SMAT files.
- 4 After locating the large files, remove them, rename and move them, or in the case of `.log` files, cycle them.

Reclaiming disk space in Windows

- 1 At the DOS prompt, change directories to `\cisversion\integrator\`.
- 2 Run the `DIR` line command. This lists files, directories, and their sizes.
- 3 Change directories to the directory that is the largest and run the `DIR` command again. This leads you to the files that are too large. These files are usually `\integrator\exec\processes\process\process.log` and `\integrator\exec\processes\process.err`.
- 4 After locating the large files, remove, rename, or move them. In the case of `.log` files, cycle them.

File permission issues

If you have a file permission problem, then use these suggested settings:

- All files in the `/integrator` file system are "664" (`rw_rw_r_`).
- Directories should be set to "776" (`rw_xrwxr_w_`).
- The `umask` is set to "002".

In UNIX, see your system administrator to change these settings, unless you have administrator privileges.

Individual and Group ownership should be set to "hci:staff". These might have been changed by the system administrator for your organization.

In Windows, all permissions are set by and can only be changed by your system administrator.

Running Cloverleaf with UAC enabled on Windows

With User Account Control (UAC) enabled, the Cloverleaf GUI can be started successfully in local and remote mode.

In some situations, the current user is a domain user in the administrator group, and using `setroot` or installing a patch does not work. In these instances, you can “Run as administrator” for Cloverleaf to successfully run.

Processes run without status information

This is usually a problem with an errant process ID. Your process directory holds the `pid` file, which holds the actual process ID that is created upon engine startup.

Check to see if the number inside the file is an actual running process.

Removing errant process ID files in UNIX

- 1 Run the `ps -ef | grep pid#` command. This command lists the process, if the process ID is running.
- 2 If the `pid` is errant, then remove the `pid` file from the process directory and restart the process.
You can also run `hcisitecleanup -p process`.

Removing errant process ID files in Windows

- 1 Open Task Manager.
- 2 Locate the `pid` file and delete it.

Restarting the engine after a power outage

- 1 From the Shell Window, set your root and site. For example, `setroot, setsite sitename`.
- 2 Run the `hcisitecleanup -p processname` command.
 - If the process is not alive, then the utility removes the `pid` and `cmd_port` files. It then checks to see if `hcimonitor` is running.
 - If `hcimonitor` is not running, then the utility runs `hcimsiutil -R`, removes the `hcimonitor pid` file and `monitorShmemFile`.
 - Then, it checks to see if the lock manager is running. If the lock manager is not running, then `hcisitecleanup` removes the lock manager `pid` file.
- 3 The engine can then be restarted.

Testing TCP/IP connections

To test your TCP/IP connections with the system, use `hcitcptest` from a command-line prompt.

```
hcitcptest [-n] [-h hostname] [-f file] -t tcp_encoding_type
-p port
```

- `-n` specifies to run in non-interactive mode, and does not read from stdin. This should be used only in server mode.
- `-h hostname` specifies an optional host name. If *hostname* is not specified, then `hcitcptest` works as a server that listens on the port that is specified by the `-p port` option. Use this argument to configure as server or client.
- `-f file` specifies an optional file name. All data received from host is written to this file, including any encoding characters.
- `-t tcp_encoding_type` specifies encoding type:
 - `2` specifies 2-byte length-encoding.
 - `2e` specifies 2-byte length-encoding, exclusive of encoding.
 - `4` specifies 4-byte length-encoding.
 - `4e` specifies 4-byte length-encoding, exclusive of encoding.
 - `orsos` specifies ORSOS 2-byte length-encoding.
 - `mlp` specifies HL7 MLP encoding.
 - `raw` specifies data is sent as-is with line feed.
 - `nlf` specifies data is sent as-is without line feed.
 - `-p port` specifies the port on which to listen or connect.

Resolving a binding error

The "unable to bind TCP/IP: connection reset by peer" error indicates that the machine with which the system was communicating has dropped its end of the connection. To resolve this:

- 1** Shut down the server side. Your organization should have procedures outlined to shut down the server side.
- 2** Shut down the client side. Your organization should have procedures outlined to shut down the client side.
- 3** Wait approximately 30 seconds and restart the server side.
- 4** Re-establish connection to the server with the client.

Client does not connect to the server

In UNIX and Windows, the host server must be is running. If it is not, then access your shell window or DOS prompt and start the host server with the `hciss -s h` command.

INEOF states

A thread that is in an INEOF state indicates that you are using the File protocol, and the thread has read all of the messages.

db_vista -921 errors

If you get a db_vista -921 error, then there are two solutions:

One solution is to run the `hcilmclear -p processname` command.

Another solution:

- 1 Shut down your processes and site daemons
- 2 Restart everything, beginning with the site daemons.

CPU is maxed out at 90% or higher

When your CPU is maxed out at 90% or higher, and you attempt to open a GUI:

- 1 Ensure your CPU meets the minimum requirements for the system to function properly.
- 2 Check and ensure that there are no other applications running.
If there are, then stop them. Your organization should have defined shutdown processes for you to follow.

Caution: You must follow shutdown and reboot instructions that are outlined by your organization's business procedures.

- 3 If this does not resolve the issue, then reboot your machine. This should resolve any problems. If it does not, then contact Support.

Virtual memory requirements

You should configure virtual memory to at least three times the RAM in your machine.

For example, if you are running with 250MB RAM, then you should configure your virtual memory to 750MB.

"/dev/kmem not world readable" error when launching Network Monitor

If you receive a "/dev/kmem not world readable" error when attempting to launch the Network Monitor, then:

- In UNIX, /dev/kmem is world readable for the Network Monitor to function properly.

- In Windows, this error does not apply to Windows machines.

Java errors

If you receive Java errors, then you must have the most current Java libraries from the manufacturer of your operating system. If necessary, then check with Support for a list of required Java libraries for your operating system.

Verify the appropriate patch levels, indicated in the release notes, are installed in your operating system.

Viewing the process directory log files

There are different methods of viewing the `.log` files depending on the operating system.

UNIX

Find the process directory `/cisversion/integrator/sitename/exec/processes/processname/processname.log`.

From the Shell Window, use `more`, `less`, or `cat` to view the `.log` file.

For example, `more processname.log` shows a verbose log.

Windows

Use the Database Administrator, or the DOS `type` command to view the `.log` files.

For example:

```
type name.log
```

You cannot edit the resulting file.

The `:q` command exits the view.

Viewing the recovery and the error databases

You can view both the recovery and error databases with the Database Administrator.

The line command for viewing the database records is `hcidbdump`.

- Your site must be set before running the command, so that site-specific information is given.
- If the command is run using `-h`, then it lists all of the flags for the command.

See [hcidbdump](#).

Command-line utilities

Command-line utilities are helpful tools in an emergency situation. These commands can assist in determining and resolving problems within the system.

To run command-line utilities, set your root and site.

For example, to set your root, use `setroot` or `setroot/cis6.x/integrator`.

To set your site, use `setsite testsite`.

hcicmd

With the engine running, `hcicmd -p process` starts the `hcicmd` prompt.

```
hcicmd -p process [-s site] [-t type] [-h host]
[-c cmd ...] [-v] [-w seconds] [-help]
```

- `-p process` specifies the engine process name. This is optional if `-t` is `d`.
- `-s site` specifies the site.
- `-t type` specifies the type of process:
 - `e` specifies engine. This is the default.
 - `d` specifies daemon.
- `-h host` specifies the process host machine.
- `-c cmd` runs the command.
- `-v` specifies verbose mode.
- `-w seconds` sets the time-out. The default is 30 seconds. The input value range is 1,65535.
- `-help` shows usage help message.

When the engine is running, `hcicmd -p process` brings up a `hcicmd` prompt. Use the prompt to communicate with the engine processes command thread.

Two examples of common uses for `hcicmd` are:

- `hcicmd -p test -c conn_1 pstop`

This stops the thread named `conn_1` in the process `test`.

- `hcicmd -p test -c conn_1 pstart`

This starts the thread named `conn_1` in the process `test`.

hconndump

This command shows threads and processes information from the `NetConfig` file within the site.

Caution: You must use the entire command!

```
hciconndump [-v] {-p process | conn_name...}
```

- `-v` specifies verbose dump.
- `-p process` dumps all connections in the indicated process.

You can also use:

`conn_name` names all the connections to dump.

hciconnstatus

This command shows the operational status of threads and processes.

```
hciconnstatus [{-p process | -c connection }]
```

`-p process` shows the operational status of the indicated process.

`-c connection` shows the operational status of the indicated connection.

This command does not function if the Monitor Daemon is not operational.

hcidbdump

This command queries, manipulates, and changes the message transaction database.

This database contains information that is related to messages that are in one of two states:

- Recovery: These are messages currently in transit through the engine.
- Error: These are messages that have not been delivered due to error.

Caution: Do not terminate `hcidbdump`! Termination before it is finished corrupts the database.

```
hcidbdump [-U userid] [-v] -B [tarfile]
```

or

```
hcidbdump {-r|-e} [-U userid] [-l] [-L [-x]] [-v] [-D] [-f source_system]  
[-o owner_system] [-d dest_system] [-t type] [-s state]  
[-S time] [-E time] [-M map] [-O time] [-a] [-b [eof]]  
[-F] [-n metadata_filename] [-m mid] [-c searchExp] [-C] [-R] [-w searchExp] [output_file]
```

- `-B` performs database backup.
- `tarfile` specifies the file name where the database backup is written. If no `tarfile` is specified, then `tar` is written to `$DBFPATH/cldb.tar`.
- `-r` saves message from recovery database.

- `-e` saves message from error database.
 - `-U userid` specifies the user ID to access database.
 - `-l` specifies long form output. This is ignored when `Output_file` is specified.
 - `-L` specifies ultra-long form output. Overrides `-l`.
 - `-x` gives an expanded view of flags. This is ignored unless using `-L`.
 - `-v` specifies verbose operation.
 - `-D` deletes messages after processing.
 - `-f source_system` selects messages with a specified source.
 - `-o owner_system` selects messages with a specified owner.
 - `-d dest_system` selects messages with a specified destination.
 - `-t type` selects messages with a specified type:
 - `D` specifies data.
 - `R` specifies reply.
 - `U` specifies unknown.
 - `-s state` selects messages with a specified state.
 - `-S time` selects messages at or after a specified date or time.
 - `-E time` selects messages at or before a specified date or time.
 - `-M map` applies the map to each message. Uses the `msgmapdata` command.
 - `-O time` orders output by ascending date. Time flags are:
 - `i` specifies inbound arrival time.
 - `o` specifies outbound time.
 - `r` specifies recovery time.
 - `x` specifies xlate start time.
 - `-a` appends data to output file.
 - `-b [eof]` dumps message to output file in length-encoded or end-of-file format.
 - `-b` dumps in length-encoded format.
 - `-b [eof]` dumps in end-of-file format. Argument `[eof]` is optional.
- Note:** You must provide an output file name for `-b [eof]`; otherwise, an error message results.
- `-F` deletes sent messages without prompting for confirmation.
 - `-n metadata_filename` enables metadata to be dumped from the error database into a file of the necessary format.
 - `-m mid` selects messages with specified message IDs:
 - `x.y.z` specifies only this message.
 - `x.y.z1:x.y.z2` specifies `x.y` messages `z1` through `z2`.
 - `-m` overrides all other restrictions.
- Note:** `-m` does not work with `-o`.
- `-c searchExp` supports regular expression search on error context.
 - `-c` prints only the message count.
 - `-R` is recovery mode for the recovery database.
 - `-w searchExp` supports regular expression search on message content. The search is case-sensitive, and works only for the error database.
 - `output_file` specifies the file name where messages are written. If no output file is specified, then message data is written to `stdout`.

Note: All specified constraints are ANDed together.

hciengineerun

This command starts the engine processes.

```
hciengineerun -p processname
```

`-p processname` starts the indicated process.

hcienginerestart

This command restarts engine processes, threads, and MonitorD. It supports the GUI for restarting an engine process, thread, and MonitorD.

```
hcienginerestart -d delay interval -p process [,process...] [-n]
```

or

```
hcienginerestart -d delay interval -p process [,process...]  
[-h host] [-s thread [, thread...] ] -- allElse [-n]
```

- `-d delay interval` is the unit for delay interval is seconds. The default value is **30s**, and is defined in the GUI at the site-level configuration. For a different time-out, use the command-line directly.
- `-p process [,process...]` restarts the indicated processes.
- `-h host` is the name of the remote host to run the engine on.
- `-s thread` is a comma-separated list of threads to restart.
- `--` separates the script args from engine args.
- `allElse` specifies all other arguments are passed to the engine.
- `-n` specifies to not run the engine in a service on Windows.

For `hciengine`:

```
hcienginerestart -d delay interval -p process [,process...]  
[-h host] [-s thread [,thread...] ] -- allElse
```

- If `hcienginerestart` is invoked when the process is not running, then the `hccmd` invocation to stop would fail. If it verifies that the process is not running after the delay interval has elapsed, then it attempts to start it with the remaining arguments.
- The `-p` argument can be a process list.
- If the process cannot be stopped, then it returns an error after the delay interval has elapsed.

For example, to restart a process:

```
hcienginerestart -d delay interval -p process
```

For threads:

```
hccmd -p process -c "thread_name prestart delay interval"
```

- If `hcidcmd` is invoked when the process is not running with `thread_name` `prestart`, then it does not connect. In this case, you can invoke `hciengine` with the thread start args. For example:

```
hcidcmd -p process -c thread name prestart delay interval
```

- In some instances, `hcidcmd` is called when the process is running, but the indicated thread is not. When this happens, it starts the thread after the delay interval has elapsed.
- In other instances, `hcidcmd` is called when the process is running, but the indicated thread has not stopped. When this happens, it returns an error after the delay interval has elapsed.

For site daemons:

```
hcisitectl [-f] [-h host] [{ -K | -k what }] [{ -S | -s what }]
[{ -R | -r what } -d delay interval] [-u #users] [-A args]
```

For site daemons it restarts when the `-R` or `-r` argument is used.

- If `hcisitectl` is invoked this way when the daemon is not running, then the invocation to stop fails. If it can verify that the process is not running after the delay interval has elapsed, then it still attempts to start it.
- If it is unable to stop the daemon, then it returns an error after the delay interval has elapsed.
- If using `-R` or `-r`, then the `-d` argument is required.

Examples:

- To restart the MonitorD:

```
hcisitectl -r m -d delay interval
```

- To restart the Lock Manger:

```
hcisitectl -r l -d delay interval
```

- To restart all daemons:

```
hcisitectl -R -d delay interval
```

The unit for `delay interval` is seconds. The default value is **30s**, and is defined in the GUI at the site-level configuration. For a different time-out, you can change `delay interval`.

hcienginestop

Stops an engine process.

```
hcienginestop -p process [-h host]
```

- `-p process` stops the indicated process.

- `-h host` specifies the name of the remote host on which to run.

hcilmclear

Cleans up after a crashed engine that has not disconnected from the Lock Manager properly.

```
hcilmclear -p process [-v]
```

- `-p process` specifies the name of the engine process to clean up.
- `-v` specifies verbose mode.

In most cases, this command should run automatically, but the command can also be run manually.

This command clears up database problems, such as "db_vista -921". This utility also eliminates the requirement to shut down the entire site to cycle the Lock Manager. Shutting down the entire site happens because one process crashes or does not shut down properly.

hcimsiutil

The system engine Monitor Statistics Interface (MSI) is accessed through the `hcimsiutil` command. This command provides access to the statistics describing the system runtime state, so that you can determine the runtime state of processes and threads.

```
hcimsiutil [-D] [-dh] [-dt] [-da] [-dd thread] [-pd proc] [-R] [-ur]
[-ut thread] [-X] [-xt thread] [-Z] [-zt thread]
[-zp process name] -rs
```

- `-D` enables debug output.
- `-dh` dumps region header.
- `-dt` dumps region Table of Contents.
- `-da` dumps all. This includes the region header and Table of Contents.
- `-dd thread` dumps data section for thread.
- `-pd proc` marks all threads in process as dead.
- `-R` removes the region.
- `-ur` unlocks the region control semaphore.
- `-ut thread` unlocks thread section semaphore.
- `-x` is the same as `-z`, except that no time values in the shared memory are reset.
- `-xt thread` corresponds to the exiting `-zt` option, although all values that are a time are retained.
- `-z` zeros all thread section data.
- `-zt thread` zeros specified thread section data.
- `-zp process name` clears the stats on `process name`.
- `rs` removes the semaphores only.

For example, to get section dump for thread `conn_1`, use:

```
hcimsiutil -dd conn_1
```

hcuprocstatus

Checks the status of a engine process.

```
hcuprocstatus [-p process] [-e]
```

- `-p process` specifies engine process name.
- `-e` specifies end-of-process status.

hcsitecleanup

Cleans up after a crashed engine that has not exited properly from all of its processes.

```
hcsitecleanup
```

Note: All processes must be down before running this command.

This utility checks to see if the process is alive. The exact path is `$HCISITE/exec/processes/process_name/pid` for UNIX and `%HCISITE%\exec\processes\process_name\pid` for Windows.

- If the process is not alive, then the utility removes the `pid` and `cmd_port` files. It then checks to see if `hcimonitor` is running.
- If `hcimonitor` is not running, then the utility runs `hcimsiutil -R`, removes the `hcimonitor pid` file and the `monitorShmemFile`.
- Then, it checks to see if the `lockmanager` is running. If the `lockmanager` is not running, then `hcsitecleanup` removes the `lockmanager pid` file.

The engine can then be restarted.

hcsitectl

This controls the site daemons.

```
hcsitectl [-f] [-h host] [{ -K | -k what }] [-v]
[{ -S | -s what }] [{ -R | -r what } -d delay]
[-u #users] [-A args] [-n]
```

- `-f` forces stopping of daemons.
- `-h host` is the name of the remote host.
- `-k` stops all daemon processes.

- `-k what` stops specified daemons and `what` is a comma-separated list:
 - `l` is the lock manager.
 - `m` is `hcimontord`.
- `-v` checks for duplicate `hcimontord` when starting/stopping MonitorD, preventing duplicate MonitorDs in one site.
- `-s` starts all daemon processes.
- `-s what` starts specified daemons and `what` is a comma-separated list:
 - `l` is the lock manager.
 - `m` is `hcimontord`.
- `-R` restarts all daemon processes.
- `-r what` restarts specified daemons and `what` is comma-separated list:
 - `l` is the lock manager.
 - `m` is `hcimontord`.
- `-d delay` is the time-out delay when using `-R` or `-r`. See [hciengine restart](#).
- `-u #users` specifies the maximum number of users for lock manager. `#users` defaults to 500 and must be greater than 200.
- `-A args` specifies the startup args for daemons and `v` is a comma-separated list.
Each entry is of the form: `what=args`, where `what` is `d`, `l`, or `m` and `args` are the arguments to pass to that daemon.
- `-n` indicates not to run under service on NT.

If no options are specified, then the status of the daemons is verified and reported.

Note: Do not stop the Lock Manager without first ensuring that there are no running engine processes in the site.

As the `errdb` alert type requires database access, during start-up the Lock Manager should be started before starting the Monitor Daemon.

For shutdown, the Monitor Daemon should be shut down first, followed by the Lock Manager.

You can also start a specific daemon with an alert. In this example, the alert file is named `foo.alert`.

```
hcisitectl -s m -A "m=-cl foo.alert"
```

MonitorD parameters

For MonitorD, the available parameters are the same as `hcimontord`:

- `-de eoc-pattern` enables an engine output configuration pattern.
- `-dd eoc-pattern` disables an engine output configuration pattern.
- `-cl cfg-list` specifies a list of alert configuration files to process. Use a space to separate multiple file names in the list. If no file name is specified, then `default.alert` is processed, if it exists.
- `-nl` enables non-daemon mode. There is no log file. In this case, the Monitor Daemon runs in the foreground. This option is only used for debugging.
- `-D` enables debugging information.
- `-S sitename` is the site name.

To run a specific alert, you can use this command. In this example, the alert file is named `foo.alert`:

```
hcisitectl -s a -A "a=-cl foo.alert"
```

hcitctest

Helps debug TCP/IP interfaces.

```
hcitctest [-n] [-h hostname] [-f file] -t tcp_encoding_type  
-p port
```

- `-n` specifies to run in non-interactive mode, and does not read from `stdin`. This should be used only in server mode.
- `-h hostname` specifies an optional host name. If `hostname` is not specified, then `hcitctest` works as a server that listens on the port that is specified by the `-p port` option. Use this argument to configure as server or client.
- `-f file` specifies an optional file name. All data received from the host is written to this file, including any encoding characters.
- `-t tcp_encoding_type` specifies the encoding type:
 - `2` specifies 2-byte length-encoding.
 - `2e` specifies 2-byte length-encoding, exclusive of encoding.
 - `4` specifies 4-byte length-encoding.
 - `4e` specifies 4-byte length-encoding, exclusive of encoding.
 - `orsos` specifies ORSOS 2-byte length-encoding.
 - `mlp` specifies HL7 MLP encoding.
 - `raw` specifies data is sent as-is with line feed.
 - `nlf` specifies data is sent as-is without line feed.
- `-p port` specifies the port on which to listen or connect.

hciverify

Performs a system validation check.

```
hciverify [-R] [-W] [-hl7] [-p] [-r dir] [-s site]  
[-u user] [-v] [-w] [-x]
```

- `-R` specifies no system default root verification.
- `-W` specifies to suppress warnings.
- `-hl7` specifies to verify only HL7 environments.
- `-p` specifies to check that expected processes are running well.
- `-r dir` specifies to verify the `root` directory (`dir`) including default.
- `-s site` specifies to verify the `site` within the `root`.
- `-u user` specifies to verify the user.

- `-v` specifies verbose operation.
- `-w` specifies to make warnings fatal.
- `-x` specifies to perform intensive xlate testing.

You must `setroot` into the `root` and `site` to verify for the `-x` option to work. `hcverify` can only be run as the `root` user. If it returns an error about files that have an incorrect mode, then rerun the command with the `-F` option. The phrase "`incorrect mode`" refers to file permissions.

If you run `hcverify` without any options, then it does not display verification information except for warnings or error messages. To have it display the verify procedure, you must run `hcverify` with the `-v` (verbose) option.

Recovery database troubleshooting

The error database holds all of the messages that error out during processing. For example, states above 100: `no route for TrxId`, `parse error`, and so on.

These topics explain DB Vista errors that can be encountered and how to handle them.

Included are steps to take for resetting the database and status, and destroying the database contents.

DB Vista -921 recovery

This error happens when a DB Vista tool has logged on to the database but crashed before logging off, leaving the user permanently logged on.

Note: Pressing **Ctrl+C** when an `hcidbdump` command is running can cause this error.

To resolve the issue on UNIX, use the `lmclean -u TEST -mp` command.

On Windows, stop all processes and daemons. Restart the system from **Control Panel > Administrative Tools > Services**.

DB Vista -921 user ID check failure

This error indicates that a user meets an exception, does not exit normally, and is not cleared when opening or operating a Raima database. When a user attempts to re-open the Raima database, the -921 error is reported.

For example:

```
(-921) 'DB_VISTA user id check failed: '_hcimonitor'_'  
,  
PANIC: "(errnum > -900) || (errnum < -966)"
```

To correct this:

- 1 Run `lmclean -s` to see whether there is a user that did not exit normally.
- 2 Run `lmclean -u $username` to clear up the user.
- 3 Rerun the program in which this error is reported. For example, rerun `monitorD`, `engine`, `hcidbdump`, or other programs.

DB Vista 940 recovery

This error happens when a database file cannot be opened because another application has it open. The application must first close the database file and then the system can start.

For example, these are some applications that could have opened the file, such as a backup or anti-virus program.

These applications must be configured to ignore any site database directories.

Resetting the database and status

- 1 Shut down all processes and the Monitor Daemon.
UNIX: Stop the Lock Manager.
Windows: Stop the CIS service.
- 2 Run `hcimsiutil -R` to remove the shared memory region.
- 3 Delete the `rdm.taf` file from the `$HCISITEDIR/exec/databases` directory.
- 4 Delete `monitorShmemFile` from the `$HCISITEDIR/exec` directory.
- 5 Initialize ICL by running `hcidbinit -I`.
- 6 Rebuild the database supporting files by running `hcidbcheck -r`.
- 7 Restart the Lock Manager, Monitor Daemon, and CIS service.
- 8 Run `hcidbdump -r`.
If problems persist, then contact Support.

Resetting the database contents

- 1 Shut down all processes and the Monitor Daemon.
UNIX: Stop the Lock Manager.
Windows: Stop the CIS service.
- 2 Delete the `databases` directory in `$HCISITEDIR/exec`.
- 3 Copy the `databases` directory from the "siteProto" site template and paste it into the `$HCISITEDIR/exec` directory.
- 4 Reset these keys in `$HCISITEDIR/siteInfo` to the plain text database:

- `dbencryption=0`
- `internaldbkey=`
- `errordbencryption=0`
- `errordbkey=`

5 Restart the Monitor Daemon and Lock Manager.

6 Start the system connections.

Resetting the status

- 1** Shut down all processes and the Monitor Daemon.
- 2** Stop the host server.
- 3** On Windows, stop the CIS service.
- 4** Run `hcimsiutil -R` to remove the shared memory region.
- 5** Delete `monitorShmemFile` from the `$HCISITEDIR/exec` directory.
- 6** Restart the Monitor Daemon and CIS service.
- 7** Start the system connections.

Index

Special Characters

-cvtnull [960](#)
 % variables [774](#)
 \$ref [433](#)
 \$ref node [427](#)
 \$schema [433](#)
 \$xlateConfigFileName [780](#)

A

access control lists
 host name [1253](#)
 role and user [1253](#)
 special characters [1253](#)
 account exception list [1123](#)
 ACL
 configuration [1217](#)
 resources/permissions [1216](#)
 acl/role manager
 acls tab [1244](#)
 events tab [1244](#)
 importing users and groups [1249](#)
 logging on [1241](#)
 roles tab [1243](#)
 users tab [1242](#)
 action breakpoints [782](#)
 active null
 setting [774](#)
 address search [267](#)
 administrator role [1212](#)
 advanced security
 security modes [1227](#)
 advanced security
 adding a role [1248](#)
 building the system [1215](#)
 certificates [1228](#)
 deleting permission settings [1256](#)
 derby [1213](#)
 host server and clients [1214](#)
 logs [1232](#)
 making a user into a role member [1248](#)
 modifying a role [1248](#)
 modifying permission settings [1255](#)
 multiview [1256](#)
 permissions [1250](#)
 read-only permission [1254](#)
 removing a site [1235](#)
 reviewing permission settings [1255](#)
 running the system [1229](#)
 security server [1214](#)
 setting permissions [1253](#)
 starting the system [1229](#)
 stopping host server [1230](#)
 advanced security (*continued*)
 stopping security server [1230](#)
 stopping the system [1229](#)
 users and roles [1247](#)
 verifying a running server [1230](#)
 affinity [1297](#)
 alert fields [186](#)
 alerts
 alert file check [325](#)
 alert message [313](#)
 alert options [309](#)
 alert properties actions pane [306](#)
 alert properties trigger pane options [304](#)
 alert search [308](#)
 alert table [308](#)
 alert time window dialog box [322](#)
 and [293](#)
 by range specifications [323](#)
 comparing optionsNew Append/After/Before [304](#)
 configuration validation [292](#)
 configuring [292](#)
 deactivating and activating [304](#)
 default [292](#)
 disk % full [294](#)
 disk free space [296](#)
 disk IO per second [296](#)
 engine status [296](#)
 error count [296](#)
 error database [296](#)
 escalating an alert [325](#)
 exec action [307](#)
 file change [297](#)
 file size [297](#)
 forward count [298](#)
 hcialertcheck [292](#)
 hcmontord usage [331](#)
 hcsitectl usage [330](#)
 hciuser profile [307](#)
 idle cpu % [298](#)
 inbound latency [298](#)
 inbound queue depth [298](#)
 inserting variables [314](#)
 last receive [298](#)
 last send [298](#)
 last update [298](#)
 license status [299](#)
 message archiving [299](#)
 outbound latency [300](#)
 outbound queue depth [300](#)
 physical memory % free [300](#)
 physical memory free [300](#)
 post-xlate queue depth [301](#)
 pre-xlate queue depth [301](#)
 process status [301](#)

- alerts (*continued*)
 - range time window dialog box 323
 - recovery database 302
 - samples 331
 - select command dialog box 328
 - starting monitord 328
 - starting monitord from command line 329
 - starting monitord from site daemons tool 329
 - substitution characters 316
 - system cpu % 302
 - tcl 302
 - tcl action 308
 - thread held 303
 - thread status 303
 - total latency 303, 314
 - transactions/sec 303
 - user cpu % 303
 - viewing alert log file 341
 - virtual memory % free 303
 - virtual memory free 304
 - wait cpu % 304
 - alias converter details 1488
 - aliases
 - viewing aliases and standards 1488
 - allOf 427
 - allowlist
 - predefined 1127
 - user-defined 1127
 - allowlist.db 1127
 - anyOf 427
 - arbitrary percision math operations 1492
 - archivelogcycle 1362
 - AS
 - auto-healing toolset 1161
 - audit log
 - accessing 1264
 - Audit Log Viewer
 - accessing 1281
 - clide.ini 1290
 - cycle files 1289
 - cycling types 1288
 - detail panel 1284
 - enabling 1281
 - exiting 1282
 - exporting the log 1284
 - log file 1291
 - log file content flags 1291
 - log history 1288
 - log levels 1280
 - log size control 1287
 - logging on 1281
 - preference options 1285
 - searching the log 1283
 - server.ini 1289
 - size keys 1290
 - sorting log entries 1283
 - tracking activities 1288
 - using the dialog box 1282
 - audit report
 - delimited 52
 - audit server 1098
 - Auto trim log 1287
 - auto-expansion levels 80
 - auto-start
 - Active-Active deployment 1167
 - Active-Passive deployment 1167
 - Active-Test deployment 1167
 - failover 1166
 - Geo-Cluster deployment 1167
 - gui options 1160
 - autostart.cfg 42
 - await replies handling 523
- ## B
- backup
 - system 1168
 - batch standard 526
 - binary format 1258
 - BLOB 603
 - blob/clob
 - command line 883
 - testing tool support 883
 - BLOB/CLOB support 365
 - box manager
 - attributes 362
 - basics 342
 - command line 353, 364
 - comparing files 359
 - configuring the environment 347
 - conflicts 358, 360
 - creating a BOX 348
 - creating new sites 361
 - dependencies 342
 - deploying project 357
 - deploying, current site 355
 - exporting 354
 - files 362
 - gui options 343
 - gui panels 344
 - importing 354
 - metadata table 346
 - overwriting a resource 360
 - plug-in threads 353
 - ports 361
 - refreshing a BOX 351
 - renaming resources 360
 - resource tree 346
 - restoring a site 364
 - searching 345
 - Select Host dialog box 344
 - transferring 354
 - user-defined folders 345
 - using database protocols 358
 - working with resources 346
 - breakpoints 784

C

- call stack
 - NetBeans IDE 995
- catch 1302, 1555
- certificate manager
 - accessing 1265
 - exiting 1267
 - starting 1265
- certificate passwords
 - updating 1090
- certificate path 1266
- certificates
 - expiration 1264
 - securing 1261–1262
 - states 1262
 - updating 1265
 - updating CA 1265
 - viewing 1275
- certificates, generating 1194
- chaining 1312
- character map 1507
- cipher suites
 - java 1185
 - OpenSSL 1185
- CL_CONX 1535
- CL_File methods 1535
- CL_HTTPServer.ini 1533
- clapi 157
- CLASSPATH 779, 1014
- clide.ini 1290
- client 1614
- client and host reconnect 125
- client preferences
 - accessing 76
 - advanced tab 77
 - alert config options 85
 - external command tab 81
 - external tools 86
 - general tab 77
 - launch bar options 86
 - netconfig/netmonitor tab 83
 - script editor tab 86
 - smat tab 84
 - xlate config tab 80
 - xsession tab 78
- client.ini 133
- clImport 1073–1074
- CLJAVA_INIT 1014
- cljTPS 1069
- cljTrxid 1069
- cljXLStrings 1069
- clLoad 1073
- CloverEnv 1032
- Cloverleaf API 1099
- cloverleaf basics 1615
- cloverleaf debugging interface 991
- clsecurityaudit 1244
- clusters 1161
- code fragment upocs 984
- code fragments 984
- column/table names
 - db schema configurator 368
- columnsBitMask 284
- command line 216
- command-line utilities
 - hcicmd 1657
 - hcionndump 1657
 - hcionnstatus 1658
 - hcidbdump 1658
 - hciengine restart 1660
 - hciengine run 1660
 - hcienginestop 1661
 - hcilmclear 1662
 - hcimsiutil 1662
 - hciprocstatus 1663
 - hcsitecleanup 1663
 - hcsitectl 1663
 - hcitcptest 1665
 - hcverify 1665
- commands
 - remote 213
 - user-defined 215
- components
 - CIS 39
 - core 40
- Compress 1287
- connect success action 593
- Connection Dump
 - viewing 161
- CONNID 725
- CONNTIMEOUT key 692
- control commands
 - foreach2 1412
- control headers
 - outgoing ibmime reply 1463
 - outgoing ibmime request 1463
- counter commands
 - CtrCurrentValue 1412, 1469
 - CtrInitCounter 1412, 1468
 - CtrNextValue 1413, 1468
 - CtrResetValue 1413, 1469
 - destroying a counter 1413
- cpu affinity 1297
- credentials 1257
- CRLs
 - refreshing 1278
 - refreshing expired 1278
- crontab 247, 1171
- CSC RMI port 1098
- csrf token 158
- cURL 662
- customer CA 1258
- customer CA files 1257
- customization
 - boot-time 42

cvtnull [1498](#)

D

database

- recovery [1611](#)
- resetting [1588](#), [1667](#)
- resetting contents [1588](#), [1667](#)

Database Administrator

- basics [163](#)
- message processing options [164](#)

database driver

- adding, editing, deleting [102](#)

database history [106](#)

database lock [1610](#)

database schema

- configuring [774](#)

database schema configurator

- adding schema [370](#)
- editing schema [370](#)
- generating schema from SQL [369](#)
- importing schema [368](#)
- ini file [365](#)
- inserting database data [371](#)
- schema options [371](#)
- sql statements [374](#)
- synchronizing schema [368–369](#)
- updating [367](#)

database schemas [177](#)

database security [1195](#)

databases

- cleaning [173](#)
- error [174](#)
- initializing [174](#)
- recovery [174](#)

datum

- destroying [1472](#)
- objects [1472](#)

datum commands

- dataget [1414](#)
- datcreate [1414](#)
- datdestroy [1414](#)
- datlist [1414](#)
- datset [1414](#)

datum extensions

- datcreate [1473](#)
- datdestroy [1473](#)
- datget [1473](#)
- datlist [1474](#)
- datset [1474](#)
- hcidatlistreset [1474](#)

datum handles [1472](#)

datum objects

- creating [1471](#)

db vista -921

- user ID check failure [1587](#), [1666](#)

db vista 940 [1588](#), [1667](#)

dblookup [457](#), [797](#)

dblookupdestroy [457](#)

dbp module [604](#)

debug mode [781](#)

debugging [782](#)

definition sub-elements [433](#)

derby [1213](#)

derby database [1170](#)

destination threads

- querying [1529](#)

dicom

- ciphers [626](#)
- TLS options [624](#)

DICOM

- encryption [626](#)
- LDL [628](#)
- message routing [629](#)
- TRXID [628](#)

DICOM SCP [622](#)

dicom scu [523](#)

DICOM SCU [623](#)

directories

- scanning multiple [655](#)

directory

- exec subdirectories [60](#)
- hierarchy [56](#)
- site-specific subdirectories [59](#)
- subdirectories [56](#)

disk-based messages

- configuring [173](#)

disk-based queuing [1313–1314](#)

DiskQue_percent [1314](#)

distributed transaction controller. See DTC [637](#)

driver components [929](#)

DRIVERCTL [726](#)

DTC

- commands [650](#)
- configuration [643](#)
- data flow [641](#)
- debugging [650](#)
- debugging engine commands [650](#)
- dtcctx [638](#)
- general options [639](#)
- protocol properties [639](#)
- query types [646](#)
- request objects [649](#)
- staging database [644](#)
- staging database tcl API [647](#)
- state and step [653](#)
- states options [640](#)
- tcl API [648](#)
- testing tool [638](#)
- transitions options [641](#)

DTCXLATE [641](#)

E

EBCDIC encoding

- searching [274](#)

- emergency save steps [124](#)
- Enable case insensitivity for user certificate authentication [1260](#)
- Enable certificate expiration notification [1260](#)
- enable Cloverleaf API [1099](#)
- Enable pending request notification [1260](#)
- encode scheme values [232](#)
- encoding
 - default [122](#)
 - fixed [963](#)
 - xml [963](#)
- encoding conversion [961](#), [969](#)
- encryption
 - DER method [1202](#)
 - dicom [626](#)
 - PEM method [1202](#)
- encryption algorithms [1201](#)
- encryption keys
 - configuring [1154](#)
- engine monitor dialog box
 - graphs menu [1647](#)
 - interval menu [1648](#)
 - totals pane [1648](#)
- engine monitor statistics interface [1515](#)
- engine output
 - configurations [117](#)
- engine output configurator
 - adding entries [383](#)
 - applying aliases [384](#)
 - basics [375](#)
 - editing entries [383](#)
 - eo alias [376](#)
 - log levels [377](#)
 - module editor [378](#)
 - modules [378](#)
 - modules and submodules [376](#)
 - selecting modules [383](#)
 - viewing output [384](#)
- English regional language setting [967](#)
- environment [1088](#)
- environments
 - selecting [1615](#)
- ephemeral ports [94](#)
- error database
 - states [1583](#)
- error database reference
 - resetting database [1588](#), [1667](#)
 - resetting database contents [1588](#), [1667](#)
 - resetting status [1589](#), [1668](#)
- error database states
 - state 107 [1585](#)
 - state 416 [1586](#)
- Error Database tool
 - basics [175](#)
 - command line usage [176](#)
 - error trap TPS [179](#)
 - schemas [177](#)
 - tcl keyed list [176](#)

- error states [168](#)
- errors
 - compilation [1302](#)
- exported server address [1093](#)
- external interfaces
 - verifying [911](#)

F

- FHIR [424](#)
- field aliases [270](#)
- file lock [39](#)
- file transfer
 - moving folders/files [123](#)
 - renaming folders/files [123](#)
 - selecting environment [123](#)
 - transferring data [123](#)
- files
 - cycling [1169](#)
- files commands
 - statfs [1415](#)
- Fileset protocols
 - using without custom tcl procs [1608](#)
- FLAGS
 - changing [1506](#)
 - metadata field [1504](#)
- FRL configurator
 - basics [385](#)
 - configuring field subfields [387](#)
 - configuring fields [386](#)
 - configuring group fields [390](#)
 - configuring group subfields [391](#)
 - configuring mask subfields [389](#)
 - configuring masks [388](#)
 - data types [391](#)
 - fill characters [392](#)
 - groups [390](#)
- functions
 - calling [1073](#)

G

- gdbm commands
 - gdbm_close [1416](#)
 - gdbm_delete [1417](#)
 - gdbm_fetch [1417](#)
 - gdbm_firstkey [1418](#)
 - gdbm_insert [1418](#)
 - gdbm_list [1418](#)
 - gdbm_load_package [1418](#)
 - gdbm_nextkey [1419](#)
 - gdbm_open [1419](#)
 - gdbm_reorganize [1419](#)
 - gdbm_replace [1419](#)
 - gdbm_store [1420](#)
 - hcigdbmconvert [1416](#)
- generate byte order mark [964](#)

generic java driver
 process_name.ini 1009
 generic java driver
 class path 1009
 design 1007
 FAQ 1009
 thread_name.ini 1009
 GETRETURNVALUE 609
 GitHub 1080
 global variables
 tcl interface usage 399
 global variables configurator
 basics 396
 encryption 398
 file location 398
 validation 399
 globalVariables.ini 398
 graphics
 performance metrics 1649
 grm commands
 grmbulkcopy 1421
 grmcreate 1421
 grmdestroy 1424
 grmencode 1424
 grmfetch 1424
 grmlist 1425
 grmnames 1425
 grmnames \$grmld 1425
 grmreset 1426
 grmstore 1426
 grm extensions
 grmbulkcopy 1480–1481
 grmcreate 1475, 1482
 grmdestroy 1484
 grmencode 1480, 1485
 grmfetch 1476
 grmlist 1485
 grmpathinfo 1486
 grmreset 1481, 1485
 grmstore 1478, 1485
 grm subclasses 1028
 grmbulkcopy 1480
 grmcreate 480, 1475
 grmencode 1480
 grmfetch 1476
 grmreset 1481
 grmstore 1478
 groups
 querying 1529
 gvsubfile 1420
 gvsubstring 1420

H

hapi 987
 hciaccess 1614, 1623
 hcialertcheck 292
 hciauditlog 1410

hcibox 353
 hcicertmgr 1410
 hcicmd
 advanced scheduling 1331
 destination threads 1325
 message forwarding 1331
 message forwarding examples 1330
 message forwarding options 1328
 resend_db 1326
 resend_errordb 1327
 resending 1328
 sticky hold 1325
 syntax 1322
 hcicreateformatdb 243
 hcicreatesite 259
 hcicrlrefresh 1410
 hcldbcheck 1588, 1667
 hcldbconvert
 converting error database 1334
 hcldbump
 completing the recovery 1339
 thread recovery 1337
 transaction logging 1338
 turning on transaction logging 1338
 usage 171
 hcldbinit 176, 514
 hcldbsetvers 176
 hcldicomdefgen 449
 hciengine 1147
 hciguitcptest 912
 hcilichelp 53
 hcilmclear 1655
 hcimonitord 216, 329, 1147
 hcimsgarchive
 concurrent archiving 1360
 data insertion template 1361
 empty list 1359
 external sql db templates 1360
 log files 1360
 marap upoc interface 1356
 marp template 1358
 multiple keyed lists 1360
 sample return values 1359
 single keyed list 1359
 siteinfo 1362
 specifying upoc 1358
 table creation template 1361
 template fields 1361
 upoc special characters 1359
 hcimsgtrace 507, 1362
 hcimsutil
 memory reset 1363
 hcinetdiff.htc 507
 hcipasswd 1410
 hciprocstatus 1369
 hciroutcopy 260, 610, 1372
 hciroutetest
 tcl procs 1374

- hcisetenv
 - clgui option [1379](#)
- hcisitecleanup [1653](#)
- hcisitetcl
 - running order [1381](#)
 - starting all daemons [1381](#)
 - starting specific alert [1382](#)
 - starting specific daemons [1381](#)
- hcisitedoc [748](#)
- hcisiteinit [259](#), [1088](#)
- hcisitetcl [1620](#)
- hcismatconvert
 - backward conversion [1385](#)
 - file-based smat to db-based smat [1386](#)
 - forward conversion [1385](#)
 - smat database file to flat file [1386](#)
 - smat db file [1385](#)
- hcismatcrypt [259–260](#)
- hciss
 - auto-start [1611](#)
 - monitoring a server [1388](#)
 - starting a server [1388](#)
 - stopping a server [1388](#)
- hcitcl [307](#)
- hcitcptest [911](#), [1654](#)
- hcitptest
 - contexts [1392](#)
- hciunixservice [1611](#)
- hciX12splitinterchange [1539](#)
- hcix12test [900](#)
- hcixlttest [781](#), [905](#)
- hcixmlcompile [866](#)
- hcixslttest
 - validation [1404](#)
- heap size [1097](#)
- helper classes
 - DispositionList [1025](#)
 - exceptions [1030](#)
 - grm [1028](#)
 - message class [1024](#)
 - MsgDataInputStream [1030](#)
 - MsgDataOutputStream [1030](#)
 - MsgDataReader [1030](#)
 - MsgDataWriter [1030](#)
 - PropertyTree [1029](#)
 - xpm [1029](#)
- High Availability
 - configuration location [1158](#)
- high availability, See AS [1159](#)
- HIPAA
 - american dental association code [919](#)
 - claim adjustment reason code [920](#)
 - claim frequency type code [920](#)
 - countries, currencies, funds [914](#)
 - CPT code [916](#)
 - external code sets [914](#)
 - health care claim status code [925](#)
 - health care financing administration plan ID [926](#)
- HIPAA (*continued*)
 - Health care procedural coding [915](#)
 - home infusion EDI coalition code [925](#)
 - ICD-9-CM procedure [916](#)
 - NAIC code [920](#)
 - national drug code [917](#)
 - national drug code by format [920](#)
 - NUBC code [916](#)
 - provider taxonomy code [926](#)
 - states [914](#)
- HIPAA external code sets [457](#)
- HL7
 - message types [1541](#)
 - segment IDs [1544](#)
 - type codes [1549](#)
- HL7 configurator
 - accessing patient name fields [405](#)
 - basics [402](#)
 - CCD support [402](#)
 - defining fields [407](#)
 - defining messages [409](#)
 - defining segments [408](#)
 - message type fields [405](#)
 - messages [406](#)
 - paths [404](#)
 - selecting fields [407](#)
 - selecting messages [409](#)
 - selecting segments [408](#)
 - standard [403](#)
- hl7 scripts
 - msgExtract [1591](#)
 - msgParser [1597](#)
 - msgReader [1592–1593](#)
- HL7 standard [403](#)
- host server
 - installing as a UNIX service [1611](#)
- host server certificate password
 - updating [1090](#)
- host server certificates
 - issuing [1273](#)
- host server/security server
 - updating certificate passwords [1091](#)
- host stats database schema [108](#)
- hostserverservice [1611](#)
- HPRIM configurator
 - basics [411](#)
 - changing segment lengths [412](#)
 - combining and splitting segments [412](#)
 - data types [414](#)
 - fields [413](#)
 - file storage [411](#)
 - messages [416](#)
 - segments [415](#)
 - separators [412](#)
- HRL configurator
 - basics [418](#)
 - condition options [419](#)
 - empty segment handling [422](#)

HRL configurator (*continued*)
 repeat options [419](#)
 http commands
 http tcl procs [1450](#)
 httpDirParse [1450](#)
 httpFilesetFetch [1450](#), [1452](#)
 httpget [1447](#)
 httppost [1446](#)
 httpput [1449](#)
 httpQuery [1451–1452](#)
 http header [1174](#)
 http headers [1175](#)
 http server interface
 CL_File [1532](#)
 CL_Tcp [1531](#)
 script inputs [1532](#)
 server-side configuration [1533](#)
 http server usage
 CL_File [1535](#)
 formats [1535](#)
 tagged format [1534](#)
 web browsers and forms [1534](#)
 httpDirParse [1450](#)
 httpFilesetFetch [1450](#), [1452](#)
 httpQuery [1451–1452](#)

I

IB client outbound
 error conditions [1464](#)
 normal message flow [1463](#)
 IB server inbound
 error conditions [1465](#)
 normal message flow [1465](#)
 IPEndPointURL [1466](#)
 ibmime commands
 proc IbClientEnvelopeOut [1460](#)
 proc IbEnvelopeIn [1459](#)
 proc IbIn [1459](#)
 proc IbInXML [1459](#)
 proc IbOut [1458](#)
 proc IbOutXML [1459](#)
 proc IbServerEnvelopeOut [1461](#)
 ibmime headers [1466](#)
 IBSOAPAction [1466](#)
 IBSOapSendTimeout [1466](#)
 icl (interthread communications library) [1299](#)
 ide
 components [63](#)
 read-only [65](#)
 searching [64](#)
 IDE [1614–1615](#)
 incoming records
 translating to outgoing [752](#)
 index notation option [753](#)
 ini and pni entries [1044](#)
 ini/pni entries
 driver section [1046](#)

ini/pni entries (*continued*)
 java driver library [1053](#)
 jvm section [1051](#)
 jvmpath section [1053](#)
 Integrated Development Environment, See IDE [1614](#)
 interface extensions [1527](#)
 internationalization [1487](#)
 interthread stats database schema [108](#)
 issues
 client not connecting to server [1654](#)
 cloverleaf and UAC [1653](#)
 contacting support [1651](#)
 cpu maxed out [1655](#)
 db_vista -921 errors [1655](#)
 dev/kmem not world readable error [1655](#)
 disk space low [1652](#)
 file permissions [1652](#)
 hcisitecleanup [1653](#)
 INEOF states [1655](#)
 java errors [1656](#)
 processes without status info [1653](#)
 resolving binding errors [1654](#)
 restarting the engine [1653](#)
 testing tcp/ip connections [1654](#)
 viewing process directory log files [1656](#)
 viewing recovery/error databases [1656](#)
 virtual memory requirements [1655](#)

J

jar files [1070](#)
 java classes
 importing [1074](#)
 java driver development kit, See jddk [1036](#)
 Java RMI [1092](#)
 Java UPoC
 debugging [996](#)
 java usage
 Bouncefile and Bounce, pattern 1 [1055](#)
 callbackJava/Bounce, pattern 3.b [1063](#)
 CallMeBack.java [1064](#)
 javadoc [1054](#)
 MessageServer, pattern 2.a [1059](#)
 MessageServer, pattern 2.b [1060](#)
 patterns [1053](#)
 RequestServer 3.b unique callback [1061](#)
 RequestServer, pattern 3.a [1061](#)
 RequestServer, pattern 3.c [1057](#)
 sample applications [1054](#)
 WeatherClient [1055](#)
 java/python/javascript class
 calling [779](#)
 javadriver threads
 multiple [1006–1007](#)
 javascript [1069](#)
 JavaScript UPoC
 debugging [996](#)

- jddk
 - class chooser dialog box [1042](#)
 - configuration [1039](#)
 - debug example [1037](#)
 - driver library [1041](#)
 - environment variables [1041](#)
 - example process [1037](#)
 - external interfaces [1039](#)
 - java driver protocol properties dialog box [1042](#)
 - message queue [1043](#)
 - methods [1042](#)
 - release compatibility [1041](#)
 - retrieving messages from engine [1040](#)
 - sending messages to engine [1040](#)
 - skip jvm shutdown hooks [1043](#)
 - sleep time [1043](#)
 - thread properties [1041](#)
 - time method options [1043](#)
 - using LinkClass [1042](#)
 - validation [1044](#)
 - war files [1038](#)
 - JKS certificates
 - issuing [1272](#)
 - JNI interface
 - calls from the driver [1066](#)
 - calls to the driver [1067](#)
 - development procedure [1068](#)
 - minimizing JVMs [1067](#)
 - joinX12
 - metadata setup [1537](#)
 - X12MetaData [1537](#)
 - JSON
 - configuration nodes [426](#)
 - JSON configurator
 - basics [424](#)
 - configuring [430](#)
 - configuring translations [431](#)
 - creating definitions [430](#)
 - FHIR [424](#)
 - manually editing nodes [430](#)
 - ordering option [428](#)
 - required nodes [428](#)
 - testing messages [431](#)
 - TrxID determination [433](#)
 - using list and tuple [427](#)
 - validation option [428](#)
 - json definition [433](#)
 - JSON nodes
 - Type and \$ref [426](#)
 - json root [433](#)
 - json schema [433](#)
 - json schema file [433](#)
 - Jython [1074](#)
- K**
- keylget [1517](#)
 - keys
 - LANG, FILE, FUNC, SCRIPT [1069](#)
 - keystore [1273](#)
- L**
- launch bar tools [1123](#)
 - LDAP
 - configuring authentication [1119](#)
 - troubleshooting [1152](#)
 - LDAP parameters [1150](#)
 - LDL configurator
 - basics [439](#)
 - configuring fields [440](#)
 - configuring messages [442](#)
 - configuring segments [441](#)
 - dcmk logging [446](#)
 - definition files [440](#)
 - DICOM and HL7 similarities [445](#)
 - DICOM generate tool [449](#)
 - handling large messages [449](#)
 - IOD [444](#)
 - logging and debugging [446](#)
 - message variation limitations [443](#)
 - retired fields [441](#)
 - ldl formats [753](#)
 - license help report contents [53](#)
 - license help tool [50](#)
 - licensing
 - alternate method [1258](#)
 - preferred method [1257](#)
 - Linux language options [968](#)
 - lists commands
 - compare [1427](#)
 - lregex [1427](#)
 - qlsort [1427](#)
 - lmclean [1587](#), [1666](#)
 - local binding [1111](#)
 - lock manager [1616](#)
 - lockmgr [216](#)
 - log history
 - configuring [1294](#)
 - features [1293](#)
 - log history setting [90](#)
 - Lookup Table configurator
 - basics [450](#), [460](#)
 - bi-directional [461](#)
 - configuration requirements [455](#)
 - configuring advanced SQL database lookup [453](#)
 - configuring basic database lookup [453](#)
 - database definition [451](#)
 - encryption [461](#)
 - error handling [455](#)
 - HIPAA external code sets [457](#)
 - masked fields [460](#)
 - multiple rows [456](#)
 - reserved table names [461](#)
 - sorting and searching table items [458](#)

Lookup Table configurator (*continued*)
 stored procedure database lookup [454](#)
 table lookup definition [450](#)
 table security [458](#)
 tcl extension [457](#)

M

mapList [1508](#)
 master site
 shared configurations [121](#)
 Maximum number of log entries [1287](#)
 message archive [1362](#)
 message class [1024](#)
 message commands
 adddatamap [1428](#)
 meggetdat [1431](#)
 msgappend [1429](#)
 msgappread [1429](#)
 msgcopy [1429](#)
 msgcreate [1430](#)
 msgdestroy [1430](#)
 msgdump [1430](#)
 msgerror [1430](#)
 msgfindchar [1431](#)
 msgfinduchar [1431](#)
 msgget [1431](#)
 msginsert [1432](#)
 msginsread [1432](#)
 msglength [1432](#)
 msglist [1432](#)
 msglock [1433](#)
 msgmapdata [1433](#)
 msgmetaget [1433](#)
 msgmetaset [1433](#)
 msgread [1434](#)
 msgrouteget [1434](#)
 msgrouteset [1434](#)
 msgset [1435](#)
 msgstats [1435](#)
 msgwrite [1435](#)
 msgXSLT [1435](#), [1553](#)
 message extensions
 adding data to a message [1499](#)
 creating message objects [1494](#)
 destroying message objects [1497](#)
 duplicating a message [1507](#)
 FLAGS field [1506](#)
 FLAGS metadata field [1504](#)
 listing message handles [1498](#)
 locking message objects [1497](#)
 managing metadata fields [1503](#)
 message transliteration [1507](#)
 metadata [1503](#)
 modifying data [1500](#)
 modifying message data [1499](#)
 modifying read-write fields [1504](#)
 msgmapdata [1507](#)

message extensions (*continued*)
 null characters [1498](#)
 querying message data [1498](#)
 reading message data [1501](#)
 specifying a key [1504](#)
 tbllookup [1508](#)
 writing message data [1502](#)
 message flags
 recovery database [1313](#)
 message flow
 inbound protocol [1310](#)
 outbound protocol [1311](#)
 xlate thread [1311](#)
 message format types [756](#)
 message movement
 interprocess [1314](#)
 message objects [1501](#)
 message references
 adddatamap [1508](#)
 msgappend [1508](#)
 msgcopy [1509](#)
 msgcreate [1509](#)
 msgdestroy [1509](#)
 msgdump [1510](#)
 msgerror [1510](#)
 msgfindchar [1510](#)
 msgfinduchar [1510](#)
 msgget [1511](#)
 msggetdat [1511](#)
 msginsert [1511](#)
 msginsread [1511](#)
 msglength [1512](#)
 msglist [1512](#)
 msglock [1512](#)
 msgmapdata [1512](#)
 msgmetaget [1513](#)
 msgmetaset [1513](#)
 msgread [1513](#)
 msgrouteget [1514](#)
 msgrouteset [1514](#)
 msgset [1514](#)
 msgstats [1514](#)
 msgwrite [1514](#)
 message states [168](#)
 messages
 deleting a single message [1609](#)
 finding next messages to be sent [207](#)
 metadata fields
 limitations [1506](#)
 metadata flags [1503](#)
 msgmetaset [272](#)
 migration
 SMAT database files [260](#)
 monitor daemon [1616](#)–[1617](#)
 monitorD [1663](#)
 monitorShmemFile [1588](#)–[1589](#), [1667](#)–[1668](#)
 mpexpr [1492](#)
 MQS SSL/TLS properties [714](#)

- msgappend [1499](#)
- msgcopy [1507](#)
- msgcreate [1494](#)
- MsgDataOutputStream [1030](#)
- MsgDataInputStream [1030](#)
- MsgDataReader [1030](#)
- MsgDataWriter [1030](#)
- msgdestroy [1497](#)
- msgdump [1503](#)
- msgfinduchar [958](#)
- msgget [1498](#)
- msginsert [1499](#)
- msglist [1498](#)
- msglock [1497](#)
- msgmapdata [1507](#)
- msgmetaget [710](#)
- msgmetaset [710](#), [1504](#)
- msgParser [1597](#)
- msgReader [1593](#)
- msgrouteget [1525](#)
- msgrouteset [1525](#)
- msgset [1500](#)
- msgwrite [1502](#)
- msi
 - initialization [1515](#)
 - modifying variable names [1516](#)
 - msiAttach [1517](#)
 - msiGetStatSample [1517](#)
 - msiTocEntry [1517](#)
 - querying table of contents [1515](#)
 - querying thread statistics [1516](#)
 - storing ToC data [1516](#)
 - testing for TOC [1516](#)
- msi commands
 - msiAttach [1436](#)
 - msiGetStatSample [1436](#)
 - msiTocEntry [1436](#)
- msiTocEntry [1516](#)
- multi-byte encoding
 - backward compatibility [962](#)
 - command line [959](#)
 - configuration tips [966](#)
 - cvtnull [960](#)
 - encoding conversion [969](#)
 - fixed encoding [963](#)
 - generate byte order mark [964](#)
 - hcicmd [959](#)
 - hcltcl [960](#)
 - hclwish [960](#)
 - invalid byte streams [957](#)
 - Java UPoC [970](#)
 - Linux installation [968](#)
 - msgfinduchar [958](#)
 - non-ascii data [958](#)
 - null characters [961](#)
 - remote machines [957](#)
 - restrictions [959](#)
 - Tcl UPoC [969](#)

- multi-byte encoding (*continued*)
 - unicode limitations [956](#)
 - unsupported encodings [957](#)
 - UPoC protocol [970](#)
 - use byte order mark [963](#)
 - xml changes [964](#)
 - xml encoding [963](#)
- multiple values
 - passing in tcl fragments [1601](#)
- multiview file [1256](#)
- mvw file [1637](#)
- mysql database [113](#)

N

- Nashorn [1074](#)
- NCI extensions [1527](#)
- NCPDP F&B
 - batch types [464](#)
 - data elements [471](#)
 - data segments [472](#)
 - definition files [470](#)
 - license key [464](#)
 - messages [473](#)
 - standard [471](#)
- NCPDP telecom configurator
 - basics [475](#)
 - batch standard [481](#)
 - composite data structures [484](#)
 - driver control keys [481](#)
 - field identifiers [481](#)
 - fields [483](#)
 - GRM tcl extensions [480](#)
 - message structure [476](#)
 - message syntax [478](#)
 - segments [480](#)
 - tokens [481](#), [484](#)
 - transactions [485](#)
 - TrxID determination [479](#)
- NetBeans
 - running [993](#)
- netconfig
 - action args [1530](#)
 - detecting modifications [1529](#)
 - loading [1528](#)
 - NCI reference [1530](#)
 - retrieving file version [1529](#)
- network configuration
 - before implementation [48](#)
 - implementation types [47](#)
 - system configuration [49](#)
- Network configurator
 - adding views [569](#)
 - basics [488](#)
 - clearing tracing database [514](#)
 - comparing formatted files [507](#)
 - configuring Branch route details [580](#)
 - configuring chain route details [579](#)

Network configurator (*continued*)

- configuring generate route details [578](#)
- configuring raw route details [577](#)
- configuring reply route details [585](#)
- configuring reply route TrxIDs [582](#)
- configuring reply routes [582](#)
- configuring reply thread properties [582](#)
- configuring route details [566](#)
- configuring routes [561](#)
- configuring the XSD WSDL [517](#)
- configuring xlate route details [577](#)
- configuring xslt route details [578](#)
- creating routes [560](#)
- creating the HTTP inbound thread [518](#)
- creating the HTTPS inbound thread [518](#)
- customizing message archiving tables [586](#)
- defining the XSD [516](#)
- DESTCONN [514](#)
- editing operations [565](#)
- editing thread filter values [570](#)
- enabling and disabling procs [490](#)
- field routing [556](#)
- globals [555](#)
- hci static else route [571](#)
- hcmsgstrace [507](#)
- inbound pane [546](#)
- inbound replies [521](#)
- inbound tab [550](#)
- inter-site destination [573](#)
- inter-site routes [513](#)
- inter-site routing [571](#)
- JVM options [498](#)
- message archiving [585](#)
- message archiving parameters [585](#)
- message tracing [504](#)
- message tracing levels [505](#)
- message tracing table schema [505](#)
- migration [587](#)
- moving threads and routes [567](#)
- msgmetaget [573](#)
- msgmetaset [573](#)
- multi-byte order options [550](#)
- multi-threaded translations [501](#)
- multiple addresses [557](#)
- notes [558](#)
- OBMSGID [542](#)
- ORIGSOURCECONN [514](#)
- outbound pane [549](#)
- outbound replies [555](#)
- outbound tab [520](#)
- process configuration [493](#)
- properties tab [543](#)
- reply generation [524](#)
- reroutes [511](#), [562](#)
- route messages tab [558](#)
- route replies tab [557](#)
- routing to destination thread [573](#)
- Save As option [489](#)

Network configurator (*continued*)

- SAVEMSGS [515](#)
- SAVETRACING [515](#)
- saving the xsd file [517](#)
- service-oriented architecture [516](#)
- setting up cloverleaf [516](#)
- siteinfo [515](#)
- SOURCECONN [514](#)
- SRCMID [514](#)
- subthreads [554](#)
- synchronizing inter-site routing [574](#)
- thread icons [490](#)
- thread template [499](#)
- transaction summary tab [559](#)
- user interface [489](#)
- user-defined options [499](#)
- verifying properties [559](#)
- view file location [568](#)
- view filter [569](#)
- view types [568](#)
- viewing routes and threads [567](#)
- wild card routing [575](#)

network monitor

- alert list order [1629](#)
- alert notification [1629](#)
- alerts [1627](#)
- alerts list [1628](#)
- command and engine output [1625](#)
- controls [1617](#)
- delay interval [1623](#)
- logs [1623](#)
- monitord error status [1623](#)
- multiple processes [1627](#)
- process control context menu [1626](#)
- process monitor [1626](#)
- starting [1623](#)
- starting host server [1622](#)
- starting lock manager [1622](#)
- starting monitor daemon [1622](#)
- tabs [1624](#)
- view/options menus [1624](#)

Network Monitor

- alert fields [186](#)
- basics [181](#)
- configuring engine output [184](#)
- creating new views [194](#)
- editing existing views [195](#)
- grid cell menu options [189](#)
- managing all threads [196](#)
- managing individual threads [196](#)
- managing the view [193](#)
- managing threads [196](#), [199](#)
- message resend [204](#)
- performance metrics [208](#)
- process control options [191](#)
- process controls [191](#)
- process monitor options [190](#)
- releasing process messages [205](#)

Network Monitor (*continued*)

- releasing thread messages [206](#)
- reviewing performance [208](#)
- reviewing the status [210](#)
- thread bitmap [188](#)
- thread control context menu options [201](#)
- Thread Controls dialog box [201](#)
- Thread Controls dialog box options [202](#)
- view control context menu [198](#)
- view control options [197](#)
- View Controls dialog box [196](#)
- view management context menu [198](#)
- view panel [188](#)
- viewing scheduled events [200](#)

network monitor properties dialog box

- depth colors tab [1632](#)
- engine output tab [1631](#)
- general tab [1630](#)
- grid view tab [1633](#)
- protocol colors tab [1632](#)
- setting protocol status color [1633](#)

Network Monitor Properties dialog box

- options [182](#)
- setting protocol status color [185](#)

nodes and acls

- adding nodes [1239](#)
- building acls [1239](#)
- security control [1239](#)
- user- and role-level permissions [1238](#)
- users and roles [1238](#)

non-ASCII xml tags [1491](#)

notifications

- setting [1260](#)

O

OBMSGID [542](#)

odbc

- application example [1566](#)
- catch [1555](#)
- command reference [1570](#)
- commands [1570](#)
- connecting data sources [1557](#)
- data types [1556](#)
- deprecated commands [1579](#)
- driver information [1558](#)
- environment and connection handles [1557](#)
- error returns [1556](#)
- loading tcl extensions [1557](#)
- memory allocation [1556](#)
- obtaining system table information [1565](#)
- preparing sql requests [1560](#)
- retrieving results [1562](#)
- setting driver options [1559](#)
- submitting requests [1561](#)
- tcl API and C API [1555](#)
- tcl extensions [1554](#)
- terminating connections [1565](#)

odbc (*continued*)

- terminating statements [1564](#)
- using descriptors [1564](#)

oneOf [427](#)

operations

- conditional [813](#)

ordered NetConfig [490](#)

P

passwords [1263](#)

pathcopy

- addresses [828](#)
- multiple sources [828](#)
- single source [827](#)

pathcopy function [824](#)

PDL

- basics [927](#)
- built-in functions [947](#)
- compiling [952](#)
- declarative section [930](#)
- device definition [930](#)
- driver definition [930](#)
- driver structure [930](#)
- encoding choices [934](#)
- exported tcl interface [932](#)
- extended backus-naur form [931](#)
- fields [936](#)
- fixed-array parts [935](#)
- length-encoded phrase parts [938](#)
- named character classes [949](#)
- phrase definition [930](#)
- phrase-checked parts [937](#)
- phrase-constant parts [935](#)
- read operations [929](#)
- sample PDL for async driver [950](#)
- sample PDL for TCP driver [949](#)
- states [928](#)
- tcl code [932](#)
- variable-array parts [936](#)
- write operations [928](#)

pdl drivers [715](#)

PDL tcl functions

- hci_pd_open_device [contns[config]] [945](#)
- uassert expr [descr] [947](#)
- hci_pd_accept info [942](#)
- hci_pd_deliver [943](#)
- hci_pd_receive contns [939](#)
- hci_pd_report_exception code text [946](#)
- hci_pd_send phrase data contns [941](#)
- hci_pd_set_result_code code [946](#)

peer authentication [624](#)

performance

- tracking and reviewing [1646](#)

permissions

- deleting [1256](#)
- modifying [1255](#)
- setting [1253](#)

- private key path [1266](#)
- proc lbServerEnvelopeOut [1461](#)
- process [1299](#)
- process configuration [1014](#)
- process controls dialog box
 - opening [1634](#)
- process stats database schema [108](#)
- processes
 - querying [1529](#)
- PropertyTree [1029](#)
- protocol message forwarding [1312](#)
- protocol status color [185](#)
- protocols
 - adding OUT CURSOR [613](#)
 - await replies [606](#)
 - basics [589](#)
 - changing websphere MQMD field values [711](#)
 - configuring a condition [618](#)
 - configuring database inbound protocols [598](#)
 - configuring database outbound protocols [600](#)
 - configuring the server port [694](#)
 - CONNID [725](#)
 - dbp module [604](#)
 - delay connection until needed [720](#)
 - DICOM [622–623](#)
 - DICOM association/networking [632](#)
 - DICOM SCP (Service Class Provider) [622](#)
 - DICOM SCU [623](#)
 - DRIVERCTL [609](#)
 - encapsulated data type [727](#)
 - file [674](#)
 - fileset FTP [654](#)
 - fileset FTP configuration details [666](#)
 - fileset FTP configuration keys [670](#)
 - fileset FTP cURL options [662](#)
 - fileset FTP inbound configuration [667](#)
 - fileset FTP inbound parameters [655](#)
 - fileset FTP outbound configuration [668](#)
 - fileset FTP outbound file name templates [661](#)
 - fileset FTP parameters [664](#)
 - fileset FTP private key [672](#)
 - fileset FTP security [671](#)
 - fileset FTP SFTP client side [673](#)
 - fileset FTP SFTP server side [673](#)
 - fileset local parameters [677](#)
 - FTP host information parameters [664](#)
 - FTP options parameters [664](#)
 - http client [686](#)
 - http client connection time-out [692](#)
 - http client cURL options [692](#)
 - http client tcl interface [691](#)
 - http client time-out [691](#)
 - http parameters [688](#)
 - inbound protocol read success/failure [619](#)
 - inserting database schema message [603](#)
 - inserting hl7 messages [603](#)
 - inserting whole message using tcl upoc [603](#)
- protocols (*continued*)
 - inserting whole message using whole_message
 - placeholder [602](#)
 - invoking chained stored procedures [613](#)
 - invoking stored procedures [612](#)
 - IPv6 [589](#)
 - Link Async [733](#)
 - local binding address [591](#)
 - LU 6.2 appc [695](#)
 - LU 6.2 appc inbound messages [700](#)
 - LU 6.2 appc initialization [700](#)
 - LU 6.2 appc outbound messages [701](#)
 - LU 6.2 appc unix configuration [697](#)
 - LU3 parameters [695](#)
 - migration [610](#)
 - NL to CRNL conversion [677](#)
 - OUT CURSOR [613](#)
 - outbound database tcl API [607](#)
 - PDL Async [715](#)
 - pdl drivers [715](#)
 - PDL TCP/IP [717](#)
 - Prosper Async [732](#)
 - protocol properties [591](#)
 - reply generation of sql chained stored procedure [607](#)
 - reply generation of sql statement [606](#)
 - reply generation of sql stored procedure [606](#)
 - returned message format [610](#)
 - returned messages [609](#)
 - scalability [590](#)
 - socket local IP binding [669](#)
 - standard query sql statements [620](#)
 - stored procedures [610](#)
 - TCP/IP [720](#)
 - TCP/IP DRIVERCTL [726](#)
 - TCP/IP length encoding [726](#)
 - TCP/IP MLP [727](#)
 - TCP/IP multi-server [725](#)
 - TCP/IP port numbers [727](#)
 - TCP/IP socket addresses [726](#)
 - UPoC [729](#)
 - websphere MQ [702](#)
 - websphere MQ configuration details [707](#)
 - websphere MQ inqueue properties [706](#)
 - websphere MQ message identifiers [709](#)
 - websphere MQ message matching [709](#)
 - websphere MQ message metadata [711](#)
 - websphere MQ message options [712](#)
 - websphere MQ message storage [708](#)
 - websphere MQ metadata changes [710](#)
 - websphere MQ option restrictions [714](#)
 - websphere MQ outqueue properties [706](#)
 - whole_message placeholder [601](#)
 - whole_message placeholder [602](#)
- python [1069](#)
- Python UPoC
 - debugging [997](#)

Q

QDXI_CLASSPATH 1014

queries

advanced 269

basic 268

queues 1301

R

receiver ID 531

recovery database

states 1586

recovery database message flags 172

reply handling 1312

resend command 228

resend_db 256

resend_errordb 229

return value 1309

RETURN_SCHEMA 610

RETURN_TCL 610

revision control 46

RMI callback port 1092

RMI object port 1092

RMI registry port 1092, 1097

roles

administrator 1212

root preferences

data encoding tab 122

setting master site 117

rootinfo 121

route command 1112

route configuration

accessing and modifying 1525

route detail type

generate 1309

raw 1307

translate 1308

routes

generate 1524

runtime tools

shell window 216

S

saveContext 282

saved translation formats 752

scheduler

adding events 111

modes 112

scheduling

advanced 734

configuring new 737

dialog box 735

event properties 736

modifying existing 738

Network Configurator 735

scheduling (*continued*)

recurring events 736

step values 737

schema 427

Script Editor

advanced features 740

client preferences options 741

external editor 745

import script 745

proc types 742

Tcl namespace 744

Tcl procedures 740

Tcl procs 739

user-defined tcl 745

script upoc

functions 1075

script UPoC classes 1070

scripts

AS service 1161

search functions 1609

search queries 270

security

CA path/CA file 1203

interaction with command line 1195

ssl client modes 1182

ssl server modes 1183

security commands

hcicrlrefresh 1410

hcupasswd 1410

hciss 1410

security errors

audit failure 1233

client gui 1233

insufficient security permission 1234

licenses 1232

log files 1232

log-in mismatch 1233

RMI 1233

security features 1190

security modes

changing 1234

security profile 624

security server

ACLs 1178

security server certificate password

updating 1091

security server certificates

issuing 1274

security server database 1156

security settings 1202

security upgrade

after upgrading 1210

basic to advanced 1209

before upgrading 1206

changing security server 1211

downgrading security 1211

levels 1205

none to advanced 1208

- security upgrade (*continued*)
 - none to basic [1207](#)
 - passwords [1206](#)
 - removing security server [1211](#)
- security.ini [131](#), [1090](#)
- Selected Encode Format option [232](#)
- Selected Encode Scheme option [232](#)
- Selected Encode Scheme setting [230](#)
- sender ID [529](#)
- server administration
 - access permission [158](#)
 - account exception list [1123](#)
 - clapi [157](#)
 - Cloverleaf API [155](#)
 - command line [1088](#)
 - configuring encryption keys [1154](#)
 - csrf token [158](#)
 - email setting [1091](#)
 - groups [1122](#)
 - heap size [1097](#)
 - host server advanced options [1094](#)
 - host server advanced security [1089](#)
 - host server encryption tab [1118](#)
 - host server firewall options [1092](#)
 - host server general options [1092](#)
 - host server LDAP tab [1119](#)
 - host server monitor tab [1098](#)
 - host server version control tab [1124](#)
 - JVM options [1096](#)
 - launch bar tools tab [1123](#)
 - LDAP advanced configuration [1151](#)
 - LDAP advanced configuration parameters [1121](#)
 - LDAP authentication [1119](#)
 - LDAP information storage [1152](#)
 - LDAP parameters [1150](#)
 - log-out options [1097](#)
 - net address translation port [1093](#)
 - open tools option [1095](#)
 - port range [1148](#)
 - RESTful API [155](#)
 - run menu passwords [1090](#)
 - security [157](#), [1089](#)
 - security server advanced tab [1149](#)
 - security server database tab [1156](#)
 - security server encryption tab [1153](#)
 - security server firewall tab [1148](#)
 - security server LDAP tab [1150](#)
 - security server web server tab [1155](#)
 - setting the environment [1088](#)
 - sites [1088](#)
 - synchronization [1150](#)
 - system ports [1147](#)
 - tcp/ip settings [1092](#)
 - troubleshooting LDAP [1152](#)
 - updating host server/security server certificates [1091](#)
 - updating the certificate password [1090](#)
 - updating the security server password [1091](#)
 - user cert and private key files [1123](#)
- server administration (*continued*)
 - user exception list [1123](#)
 - web services [1100](#)
 - xlt debug traffic [1096](#)
 - xss attack protection [158](#)
- server interface
 - command-line tools [117](#)
- server.ini [126](#), [1089–1090](#), [1093](#), [1289](#)
- setroot [1657](#)
- setsite [1657](#)
- shared_files [122](#)
- shell commands [287](#)
- site
 - adding/deleting [1088](#)
- site daemons
 - accessing [1616](#)
 - command line usage [1620](#)
 - command pane [1618](#)
 - logfile [1293](#)
 - parameter pane [1617](#)
 - running order [1621](#)
 - selecting alerts [1618](#)
 - starting [1617](#)
 - starting specific alert [1621](#)
 - starting specific daemon [1621](#)
 - stopping [1617](#)
 - using [217](#)
 - using -S argument [1621](#)
 - viewing error log [1619](#)
 - viewing log file [1619](#)
 - viewing status [1619](#)
- site document
 - accessing [747](#)
 - basics [747](#)
 - generating [749](#)
 - generating from the gui [749](#)
 - local/remote access [750](#)
 - server interface [748](#)
- site document generated files [1099](#)
- site folders [1088](#)
- site init
 - commands [69](#)
 - creating a unix site [66](#)
 - creating a Windows site [66](#)
 - symbolic link [67](#)
- site manager
 - refresh [64](#)
 - searching [64](#)
- site preferences
 - alert config tab [97](#)
 - database config tab [99](#)
 - error handling tab [96](#)
 - external database tab [104](#)
 - general tab [88](#)
 - internal/error database tab [103](#)
 - log history tab [89](#)
 - netconfig tab [90](#)
 - smat tab [90](#)

- site preferences (*continued*)
 - statistic database tab [106](#)
 - system management tab [94](#)
- site stats database schema [108](#)
- site, renaming [1088](#)
- siteinfo file [88](#)
- siteproto directories [1013](#)
- siteSecurityInfo [1195](#)
- siteSecurityInfo keys [1200](#)
- smart search [266](#)
- SMAT
 - adding all message to view [224](#)
 - automatic SMAT cycling [236](#)
 - basics [217](#)
 - cell editors [225](#)
 - client preferences options [224](#)
 - condition list table [222](#)
 - condition list toolbar [222](#)
 - current message panel [219](#)
 - cycling [236](#)
 - date expressions [223](#)
 - destination threads [241](#)
 - editing conditions [223](#)
 - file synchronization [227](#)
 - file types [217](#)
 - fixed length fields [234](#)
 - hcicmd [228](#)
 - hcidbdump [229](#)
 - history [236](#)
 - inbound/outbound messages [231](#)
 - index files [233](#)
 - loading multiple files [220](#)
 - message encoding options [241](#)
 - message file [233](#)
 - message formats [242](#)
 - message metadata [225](#)
 - message states [218](#)
 - metadata fields [226](#)
 - monitoring parsing [243](#)
 - multi-line search [224](#)
 - process configuration [236](#)
 - purging [236](#)
 - resend_errordb [229](#)
 - resending messages [240](#)
 - resending messages with metadata [242](#)
 - restoring last view [224](#)
 - saved message file [218](#), [240](#)
 - saved message file access [240](#)
 - saved message file cleanup [240](#)
 - saved message format [233](#)
 - secondary file/database [239](#)
 - site preferences [238](#)
 - specifying destination threads [228](#)
 - temp files [228](#), [233](#)
 - using regular expressions [224](#)
 - variable length fields [235](#)
 - view definition [221](#)
- SMAT API [274](#)
- SMAT API usage [285](#)
- SMAT database
 - advanced criteria [245](#)
 - changing incorrect keys [263](#)
 - changing the key [262](#)
 - cycling and closing [287](#)
 - deleting messages [253](#)
 - deleting old messages [266](#)
 - encryption [256](#)
 - encryption key [258](#)
 - external message flags [272](#)
 - hcismatconvert [257](#)
 - hcismatcrypt [259](#)
 - HL7 smart search [266](#)
 - HL7/HPRIM advanced queries [269](#)
 - HL7/HPRIM basic queries [268](#)
 - HL7/HPRIM search options [267](#)
 - HL7/HPRIM separator cache [267](#)
 - loading SMATDB key [259](#)
 - maintaining field aliases [270](#)
 - message metadata [254](#)
 - message resend [256](#)
 - migrating encrypted SMAT [261](#)
 - migrating SMAT DB files [260](#)
 - recovering data [262](#)
 - regex limitations [271](#)
 - renamed database site issues [262](#)
 - search query variations [270](#)
 - search results [252](#)
 - searching [256](#)
 - Site Init dialog box [258](#)
 - smat_msgs table columns [264](#)
 - SMATDB key [259](#)
 - sql extensions [264](#)
 - sqlite [263](#)
 - sqlite error log [260](#)
 - starting a search [243](#)
 - using sqlite [263](#)
 - using tcl script [263](#)
- SMAT database file
 - corrupt [252](#)
- SMAT message editor [225](#)
- SMAT messages
 - deleting [288](#)
- SMAT search, canceling [243](#)
- smat_msgs [264](#)
- smatdbinsert [281](#)
- soap extensions [1174](#)
- soap header [1174](#)
- soap headers, arbitrary [1176](#)
- source/destination values [771](#)
- splitX12
 - group split [1539](#)
 - group-defined transaction sets [1540](#)
 - hciX12splitinterchange [1539](#)
 - saving headers [1540](#)
 - transaction split [1538](#)
 - TrxID determination [1541](#)

- sql extensions [264](#)
- sqlite [263](#)
- SSL [1180](#)
- ssl authentication [1194](#)
- SSL certificates [1149](#)
- ssl connections [1181](#)
- ssl modes [1203](#)
- ssldap.xml [1152](#)
- staging database [647](#)
- state information [1310](#)
- state machine [641](#)
- status
 - resetting [1589](#), [1668](#)
- Step Into [994](#)
- Step Over [994](#)
- strings commands
 - strsub [1437](#)
- suspension tab [1098](#)
- symbolic link
 - creating [67](#)
 - outside HCIROOT [68](#)
 - using smat [69](#)
- synchronization parameters [1150](#)
- system administration
 - updating certificate passwords [1090](#)
- system components [1614](#)
- system maintenance
 - automating in unix [1171](#)
 - automating in windows [1172](#)
 - cycling files [1169](#)
 - database directory [1169](#)
 - file system [1169](#)
 - management tips [1171](#)
 - tcl leaks [1173](#)
 - unix systems [1170](#)
 - windows systems [1171](#)
- system tools [1614](#)
- system-level environment variables [1553](#)

T

- table commands
 - dblookup [1438](#)
 - dblookupdestroy [1438](#)
 - tbllookup [1437](#)
- Table smat_msgs definition [283](#)
- tbllookup [1508](#)
- tcl [42](#), [1302](#)
- Tcl 7.6
 - upgrading to 8.X [1303](#)
- Tcl API commands [275](#)
- tcl code
 - selecting [779](#)
- Tcl commands [286](#)
- Tcl evaluation [1032](#)
- tcl help [1602](#)
- tcl interpreter [1302](#)
- Tcl interpreter [1490](#)
- tcl log [88](#)
- Tcl plug-in
 - installing [993](#)
- Tcl SMAT API
 - default saveContext [282](#)
 - disposition list [279](#)
 - resend and trace [281](#)
- Tcl source files [994](#)
- Tcl string map [1605](#)
- Tcl syntax [286](#)
- Tcl templates [1466](#)
- Tcl UPoC
 - debugging [992](#)
- TCP/IP
 - testing connections [912](#)
- tcp/ip client threads
 - configuring outbound [1608](#)
- tcp/ip server threads
 - configuring inbound [1607](#)
- Testing Tool
 - database protocol [886](#)
 - database schema [887](#)
 - DICOM [888](#)
 - EDIFACT [889](#)
 - FRL [890](#)
 - grep [885](#)
 - HL7 [890](#)
 - HPRIM [891](#)
 - HRL [892](#)
 - integration and testing [885](#)
 - JSON [893](#)
 - LDL [894](#)
 - lookup table extension [883](#)
 - NCPDP F&B [904](#)
 - NCPDP SCRIPT [903](#)
 - NCPDP Telecom [902](#)
 - routes [895](#)
 - system tests [884](#)
 - system topology [884](#)
 - test output [885](#)
 - TPS [897](#)
 - TRX ID [899](#)
 - unit testing [884](#)
 - VRL [900](#)
 - X12 [900](#)
 - XLT [905](#)
 - XML [906](#)
 - XSLT [907](#)
 - XSLT files [909](#)
- thread
 - command [41](#)
 - definition [41](#)
 - protocol [41](#)
 - translation [41](#)
- thread bitmaps [188](#)
- thread control context menu
 - options [1643](#)

- thread control dialog box
 - opening [1644](#)
 - options [1644](#)
- thread management
 - viewing scheduled events [1643](#)
- thread metrics
 - reviewing [1647](#)
- thread stats database schema [108](#)
- threads
 - querying [1528](#)
- threshold in megabytes option [173](#)
- tilde [773](#)
- timer thread state [738](#)
- tps
 - arguments [973](#)
 - contexts [974](#)
- TPS
 - action [653](#)
 - rollback [654](#)
 - script upoc [1075](#)
- TPS message flow [179](#)
- tps module [973](#)
- TPS module interface [976](#)
- TPS test tool [1014](#)
- translation actions
 - creating [753](#), [790](#)
- translation configurator
 - action properties [816](#)
 - applying action properties [817](#)
 - auto entry [801](#)
 - basics [750](#)
 - basis customization [801](#)
 - BREAK action [815](#)
 - BULKCOPY action [808](#)
 - CALL action [811](#)
 - COMMENT action [808](#)
 - CONCAT action [794](#)
 - configuring action properties [816](#)
 - CONTINUE action [811](#)
 - COPY action [791](#)
 - copying/pasting [777](#)
 - data bundles of arbitrary size [809](#)
 - database table [795](#)
 - DATECOPYOPT action [811](#)
 - DATECOPYOPT missing centuries [811](#)
 - DATECOPYOPT timestamp [812](#)
 - DATECOPYOPY time options [812](#)
 - dblookup [797](#)
 - error handling [777](#)
 - expanding operate statements [818](#)
 - fetching value [791](#)
 - hardcoding literal values [772](#)
 - IF action [812](#)
 - INCLUDE action [803](#)
 - INCLUDE action in xlate debugger [805](#)
 - inserting BREAK in IF action [816](#)
 - ITERATE action [798](#)
 - ITERATE type user [800](#)

- translation configurator (*continued*)
 - ITERATE types [798](#)
 - ITERATE variables [799](#)
 - list ITERATE values [799](#)
 - MATH action [806](#)
 - message flow [771](#)
 - multiple copied actions [817](#)
 - parsing [752](#)
 - PATHCOPY action [809](#)
 - searching [776](#)
 - selecting formats [751](#)
 - SEND action [815](#)
 - source message view [785](#)
 - source/destination selection [771](#)
 - STRING action [806](#)
 - SUPPRESS action [815](#)
 - TABLE action [795](#)
 - tbl file [796](#)
 - using INCLUDE [804](#)
 - xml support [775](#)
- translation formats
 - saving [753](#)
- translation operators [813](#)
- trlogging [1338](#)
- Trxid
 - script upoc [1076](#)
- trxID determination [985](#)
- type substitution [761](#)

U

- UltraLong
 - using [1609](#)
- UN/EDIFACT configurator
 - composite data structures [834](#)
 - data elements [834](#)
 - data segments [835](#)
 - encoding separators [832](#)
 - message formats [832](#)
 - messages [836](#)
 - SEPCARS metadata field [833](#)
 - tps procedures [833](#)
- UNIX users
 - setting up [1610](#)
- Update All Hashes button [1126](#)
- upoc java
 - architecture [1004](#)
 - calling java tps from tcl [1033](#)
 - calling java TrxID from tcl [1034](#)
 - calling java xlt from tcl [1035](#)
 - calls from the driver [1066](#)
 - calls to the driver [1067](#)
 - code fragment interface [1015](#)
 - constructors and userArgs [1004](#)
 - destroy method [1005](#)
 - development procedure [1068](#)
 - directories [1013](#)
 - environment setup [1014](#)

- upoc java (*continued*)
 - helper classes [1024](#)
 - ini and pni entries [1044](#)
 - instances and construction [1005](#)
 - JNI interface [1066](#)
 - jvm auto defines [1044](#)
 - logging in java driver [1068](#)
 - minimizing JVMs [1067](#)
 - static variables [1005](#)
 - subclasses [1015](#)
 - thread template [1044](#)
 - tps module interface [1019](#)
 - trxID determination interface [1014](#)
- upoc subclasses [1015](#)
- upocs
 - script [1069](#), [1074](#)
 - TPS module interface [976](#)
- use byte order mark [963](#)
- Use byte order mark [955](#)
- use credential cache [1266](#)
- user cache tab [1098](#)
- user certificates
 - issuing with user participation [1268](#)
 - issuing without user participation [1269](#)
 - requesting [1271](#)
 - revoking [1277](#)
 - un-revoking [1277](#)
- user exception list [1123](#)
- user iterate type [800](#)
- user passwords
 - changing [1275](#)
- users
 - adding new [1266](#)
- utf-8 [958](#)

V

- validation
 - commands [1128](#)
 - java [1129](#)
- variable names
 - % variables [774](#)
- variables
 - NetBeans IDE [994](#)
 - temporary [773](#)
- variant search [267](#)
- version control
 - comparing versions [73](#)
 - databases and folders [71](#)
 - site manager context menu [71](#)
 - statuses [74](#)
- view control context menu [198](#), [1639](#)
- view controls dialog box
 - accessing [1640](#)
 - options [1641](#)
- View Controls dialog box [196](#)
- view management [1639](#)

- views and threads
 - creating new views [1640](#)
 - editing views [1640](#)
 - managing [1636](#)
 - reply route representation [1637](#)
 - route lines [1637](#)
 - starting all threads in a view [1638](#)
 - starting individual threads [1638](#)
 - thread icons [1636](#)
 - updating views [1640](#)
 - view control context menu [1639](#)
 - view management context menu [1639](#)
 - view pane [1637](#)
 - view types [1638](#)
- VRL configurator
 - fields [841](#)
 - group fields [846](#)
 - group subfields [847](#)
 - group subsubfields [848](#)
 - mask subfields [845](#)
 - mask subsubfields [845](#)
 - masks [844](#)
 - subfields [842](#)
 - subsubfields [843](#)
- VRL definition [840](#)

W

- web server [1155](#)
- web services
 - Error message resend [1107](#)
 - SMAT message resend [1101](#)
- web services certificate
 - creating [1264](#)
- windows authentication [112](#)
- Windows language options [967](#)
- Windows multi-language [967](#)
- wizard
 - adding variants [148](#)
 - configuring localization [144](#)
 - netconfig metadata [148](#)
 - setting up [141](#)
- WS Client GUI [1174](#)
- WS Raw Client GUI [1174](#)
- WSDL2XSD [850](#)

X

- X12
 - functional ID code [532](#)
 - header options [527](#)
 - version number [541](#)
- X12 configurator
 - basics [850](#)
 - composite data structures [855](#)
 - data elements [855](#)
 - data segments [856](#)

- X12 configurator (*continued*)
 - HIPAA [854](#)
 - optional/releasing groups [857](#)
 - syntax notation [851](#)
 - transaction sets [856](#)
- X12 procedures
 - joinX12 [1537](#)
- X12MetaData [1537](#)
- xlate commands
 - ADDPREC [1440](#)
 - convert_date [1439](#)
- xlate debugger
 - action breakpoints [782](#)
 - breakpoints [784](#)
 - console view [785](#)
 - debug buttons [786](#)
 - debug panel [783](#)
 - debugging [782](#)
 - expressions view [784](#)
 - input values view [785](#)
 - source message view [785](#)
 - target message view [785](#)
 - variables view [784](#)
- xlateInVals [1600](#)
- xlateOutVals [1600](#)
- XltCall
 - script upoc [1076](#)
- XltObject
 - script upoc [1077](#)
- XltObjects
 - script upoc [1077](#)
- XltString
 - script upoc [1078](#)
- XltStrings
 - script upoc [1079](#)
- xml
 - message encoding [760](#)
- XML package manager
 - CDA documents [873](#)
 - compile options parameters [874](#)
 - compile properties [875](#)
 - compiling [871](#)
 - compiling/recompiling [866](#)
 - creating CDA packages/folders [874](#)
 - creating new folders [866](#)
 - default behavior [869](#)
 - default namespace [868](#)
 - definition files [862](#)
 - empty node pruning [871](#)
 - file management [862](#)
 - files and folders [864](#)
 - hcxmlcompile [867](#)
 - HL7 support [872](#)
 - index files [880](#)
 - moving XSD files [879](#)
 - namespaces [876](#)
 - OCM files [872](#)
 - OCM files and namespaces [877](#)
- XML package manager (*continued*)
 - output OCM file name [870](#)
 - pretty print [868](#)
 - recompiling [871](#)
 - renaming files/folders [866](#)
 - schema support [875](#)
 - schemas [874](#)
 - security [861](#)
 - Testing Tool output [876](#)
 - TrxID determination [878](#)
 - XML configuration files [864](#)
- xml path semantics
 - path with type/repetition modifiers [770](#)
 - source type modifier [769](#)
 - xsi:type changes [770](#)
 - xsi:type set order [770](#)
 - xsi:type value [768](#)
- xml pathcopy [826](#)
- xml schema wildcards
 - DTD ANY and schema anyType [767](#)
 - user interface [765](#)
- xml terms [859](#)
- xml tree structure
 - "use" property [759](#)
 - hierarchy [758](#)
 - messages [756](#)
 - path string generation [759](#)
 - recursion [759](#)
 - symbols [756](#)
 - tree icons [757](#)
 - tree view labels [757](#)
- xml type substitution
 - abstract types [763](#)
 - configuring substitutions [764](#)
 - substitution type selection [764](#)
- xpm
 - appending messages to output [1520](#)
 - architecture [1518](#)
 - error generation [1521](#)
 - extensions [1517](#)
 - handles [1518](#)
 - message encoding [1520](#)
 - metadata access [1519](#)
 - modifying field data [1519](#)
 - querying field data [1518](#)
 - xpmdispose [1521](#)
 - xpmencode [1522](#)
 - xpmerror [1522](#)
 - xpmfetch [1522](#)
 - xpmmetaget [1523](#)
 - xpmmetaset [1523](#)
 - xpmstore [1523](#)
- xpm commands
 - xpmdispose [1443](#)
 - xpmencode [1443](#)
 - xpmerror [1443](#)
 - xpmfetch [1444](#)
 - xpmmetaget [1444](#)

xpm commands (*continued*)

- xpmmetaset [1445](#)
- xpmstore [1445](#)

xpm helper classes [1029](#)
xpmdispose [1520](#)
xpmencode [1520](#)
xpmerror [1521](#)

xpmfetch [1518](#)
xpmmetaget [1519](#)
xpmmetaset [1519](#)
xpmstore [1519](#)
XSD WSDL tool [850](#)
xss attach protection [158](#)