



# Catalys Node User Guide

Version 2.2 r692de9dd

Publication date 2024-02-22T15:51:59+01:00

Copyright © 2024 CameronTec

# Table of Contents

1. Introduction .....	1
1.1. Catalys Overview .....	1
1.2. Catalys Node Overview .....	2
1.3. Catalys Releases .....	3
1.4. What's New in Catalys 2.2 .....	4
1.4.1. Installers .....	4
1.4.2. New Features in Catalys Node 2.2 .....	4
1.4.3. Public API changes Catalys Node 2.2 .....	6
1.4.4. Known Issues in Catalys Node 2.2 .....	6
1.4.5. New Features in Catalys Node 2.1 Maintenance Releases .....	7
1.4.6. New Features in Catalys MIS 2.2 .....	10
1.4.7. Known Issues in Catalys MIS 2.2 .....	10
1.4.8. New Features in Catalys MIS 2.1 Maintenance Releases .....	12
1.4.9. New Features in Catalys LMA 2.2 .....	12
1.4.10. Known Issues in Catalys LMA 2.2 .....	12
1.5. Getting Support .....	13
1.5.1. Contacting Support .....	13
1.5.2. Sending Files to Your Support Team .....	14
1.6. FAQs .....	14
2. Installation and Startup .....	20
2.1. Prerequisites .....	20
2.2. Support Portal .....	21
2.3. License Management .....	22
2.3.1. License File .....	22
2.3.2. How to Read Your License File .....	22
2.3.3. License Maintenance .....	23
2.4. New Installation .....	23
2.4.1. Introduction .....	23
2.4.2. Installation Steps .....	23
2.4.3. Post-Installation Core Node Directories .....	25
2.4.4. Post-Installation Node Instances Directories .....	25
2.4.5. Upgrading or Installing Additional Features .....	26
2.4.6. Uninstaller .....	26
2.5. Migrating a Previous Installation .....	27
2.5.1. Introduction .....	27
2.5.2. Java .....	27
2.5.3. License File .....	27
2.5.4. DTD Reference .....	27
2.5.5. Migrating Node Instances Without Using the Installer .....	28
2.5.6. Migrating Node Instances Using the Installer .....	28

2.5.7. XML Configuration .....	28
2.5.8. Logging Configuration .....	29
2.5.9. Persistent Data Files .....	29
2.5.10. High Performance Logger (HPL) Data Files .....	29
2.6. Startup .....	29
2.6.1. XML Configuration File .....	29
2.6.2. Startup Script .....	30
2.6.3. Starting the Example Node Instance .....	31
2.6.4. Troubleshooting .....	32
3. Configuration .....	34
3.1. Overview .....	34
3.1.1. Application Configuration .....	35
3.1.2. Services Configuration .....	35
3.1.3. Session Configuration .....	36
3.1.4. Connection Configuration .....	37
3.1.5. Persister Configuration .....	37
3.1.6. Session Manager Configuration .....	38
3.1.7. Instance and Session Properties Configuration .....	42
3.1.8. Message Processing Templates Configuration .....	42
3.1.9. Replacement Parameters .....	43
3.2. Logging .....	44
3.2.1. Choosing a Logging Implementation .....	44
3.2.2. HPL Configuration and Operation .....	45
3.2.3. Logback Configuration .....	48
3.2.4. FIX Message Logging .....	51
3.2.5. Logging API .....	52
3.3. JMX Configuration API .....	53
3.3.1. Introduction .....	53
3.3.2. Management Infrastructure .....	53
3.3.3. Configuration versus Runtime .....	54
3.3.4. Reload .....	54
3.3.5. Refresh .....	54
3.3.6. Undo .....	55
3.3.7. Persistence .....	55
3.3.8. MBeans .....	55
3.3.9. Interactively Exploring the API .....	56
3.3.10. Example .....	57
3.4. Encryption of Attributes .....	57
3.4.1. Encrypt an Attribute .....	58
3.4.2. Decrypt an Attribute .....	58
4. Operations and Monitoring .....	59
4.1. Node Lifecycle .....	59
4.1.1. Introduction .....	59

4.1.2. Starting the Node .....	60
4.1.3. Starting FIX Sessions .....	60
4.1.4. Stopping FIX Sessions .....	60
4.1.5. Resetting FIX Sessions .....	60
4.1.6. End-of-Day Processing .....	61
4.1.7. Stopping the Node .....	61
4.1.8. Archiving Logs .....	61
4.2. Monitoring the Node .....	61
4.2.1. Introduction .....	61
4.2.2. Command-Line Interface .....	61
4.2.3. Remote Command-Line Interface .....	70
4.2.4. Custom CLI Commands .....	73
4.2.5. Monitoring via JMX .....	73
4.2.6. Monitoring via the Dashboard .....	78
4.2.7. Developing a Custom Monitor Class .....	78
4.3. Configuration Model and Runtime Changes .....	78
4.3.1. Introduction .....	78
4.3.2. Configuration Model vs. Runtime Model .....	78
4.3.3. Dynamic Configuration .....	79
5. Standard Features .....	86
5.1. Session Persistence .....	87
5.1.1. Introduction .....	87
5.1.2. JournalingPersister .....	88
5.1.3. MemoryPersister .....	90
5.1.4. MemorySeqNoOnlyPersister .....	91
5.2. Scheduling .....	91
5.2.1. Introduction .....	91
5.2.2. Configuration Overview .....	91
5.2.3. Configuration Examples .....	95
5.2.4. Scheduling and Daylight Savings Time .....	97
5.2.5. In Schedule vs. Outside Schedule .....	98
5.2.6. Custom Tasks .....	98
5.2.7. JMX Management .....	99
5.3. FIX Routing .....	102
5.3.1. Introduction .....	102
5.3.2. Routing Modes .....	102
5.3.3. Making Routing Decisions .....	104
5.3.4. Routing Side .....	105
5.3.5. Routing Networks .....	106
5.3.6. Handling Routing Issues .....	106
5.3.7. FIX Routing Configuration .....	107
5.4. FIX Drop Copy .....	109
5.4.1. Introduction .....	109

5.4.2. Drop Copy Methods .....	110
5.5. Message Buffering .....	110
5.5.1. Introduction .....	110
5.5.2. Configuration .....	110
5.5.3. Persistent Collection Configuration .....	112
5.5.4. Managing Buffers from the Command Line .....	112
5.6. Message Throttling .....	113
5.6.1. Introduction .....	113
5.6.2. Configuration .....	113
5.7. FIX Dictionaries .....	115
5.7.1. Introduction .....	115
5.7.2. Getting Started .....	115
5.7.3. Custom Dictionaries .....	116
5.7.4. Configuration of Custom Dictionary .....	116
5.7.5. Dictionaries API .....	116
5.7.6. Non XML-based Dictionaries .....	116
5.8. Inbound Message Validation .....	117
5.8.1. Introduction .....	117
5.8.2. Validation Rules .....	117
5.9. Message Factories .....	122
5.9.1. Introduction .....	122
5.9.2. Configuration .....	122
5.9.3. Provided Implementations .....	123
5.9.4. Custom Implementation .....	124
6. Optional Features .....	125
6.1. High Availability .....	126
6.1.1. Introduction .....	126
6.1.2. Role Determination .....	127
6.1.3. Failover .....	127
6.1.4. State Persistence .....	129
6.1.5. Single IP Address .....	131
6.1.6. Configuration .....	134
6.1.7. High Availability Operations .....	139
6.1.8. High Availability Communications .....	144
6.2. Catalys Rules Engine .....	148
6.2.1. Introduction .....	148
6.2.2. Configuration .....	149
6.2.3. Expression Language .....	152
6.2.4. Conditions .....	157
6.2.5. Actions .....	162
6.2.6. Message Validation .....	177
6.3. File Lookup Service .....	178
6.3.1. Introduction .....	178

6.3.2. Basic Configuration Example .....	178
6.3.3. Configuring Multiple Keys and Results .....	179
6.3.4. Lookup from a Custom Message Processor .....	180
6.3.5. Labels .....	181
6.4. Scheduled Order Cache .....	182
6.4.1. Introduction .....	182
6.4.2. Configuration .....	182
6.4.3. Behavior of Scheduled Order Cache .....	184
6.5. FIXML Transformer .....	184
6.5.1. Introduction .....	184
6.5.2. Configuration .....	185
6.5.3. Customizing FIXML Definitions - FixmlElementGenerator .....	186
6.5.4. Templates .....	187
6.6. SSL Socket Connection .....	187
6.6.1. Introduction .....	187
6.6.2. Java Keystores and Truststores .....	188
6.6.3. SSL Protocols .....	188
6.6.4. Configuration .....	189
6.6.5. Client Authentication .....	192
6.6.6. Troubleshooting .....	194
6.7. Data Encryption .....	195
6.7.1. Introduction .....	195
6.7.2. Configuration .....	195
6.7.3. Encryption Keys .....	197
6.7.4. Troubleshooting .....	197
6.8. Breakout Box .....	198
6.8.1. Introduction .....	198
6.8.2. Concepts .....	198
6.8.3. Examples .....	199
6.8.4. Configuration .....	204
6.9. Market Compliance .....	204
6.9.1. Introduction .....	204
6.9.2. Compliance Filters .....	205
6.9.3. Operational Model .....	207
6.9.4. Configuration .....	207
6.9.5. Rules File .....	208
6.9.6. Price Bands for Percentage Filters .....	220
6.9.7. Price Step Table File .....	222
6.9.8. Extending Filters .....	222
6.9.9. The <code>absolute</code> Attribute .....	223
6.10. Message Compression .....	223
6.10.1. FAST Support .....	223
6.10.2. FAST and the FIX Libraries .....	224

6.10.3. Additional Notes .....	224
6.11. Market Data Services .....	224
6.11.1. Introduction .....	224
6.11.2. Configuring a Market Data Service .....	225
6.11.3. Using a Market Data Service .....	226
6.11.4. Writing a Market Data Service .....	226
6.11.5. Writing a Derived Market Data Service .....	227
7. Technology Adapters .....	228
7.1. Socket Adapter .....	230
7.1.1. Introduction .....	230
7.1.2. Socket Adapter Protocol .....	231
7.1.3. Configuration .....	233
7.1.4. Sample Client Code .....	234
7.1.5. Message Transformers .....	236
7.1.6. Multi-Session Support .....	236
7.1.7. Message Acknowledgements .....	237
7.2. Advanced Socket Adapter .....	237
7.2.1. Simple vs. Advanced .....	237
7.2.2. Application Architecture .....	238
7.2.3. Protocol .....	238
7.2.4. Configuration .....	251
7.2.5. Command-Line Operations .....	252
7.3. Embedded Catalys Node Adapter .....	253
7.3.1. Introduction .....	253
7.3.2. Adapter Operation .....	253
7.3.3. Configuration .....	253
7.3.4. Templates and Example Client Code .....	254
7.3.5. Performance .....	256
7.4. JMS Adapter .....	256
7.4.1. Introduction .....	256
7.4.2. JMS Provider Installation .....	256
7.4.3. Configuration .....	257
7.4.4. Templates .....	258
7.5. Solace Adapter .....	258
7.5.1. Introduction .....	258
7.5.2. Installation .....	258
7.5.3. Functionality .....	259
7.5.4. Solace Adapter Components .....	259
7.5.5. Configuration .....	262
7.5.6. Performance Tuning .....	264
7.5.7. Using the Adapter with Catalys Node High Availability .....	265
7.5.8. JCSMP API Logging .....	266
7.6. IBM MQSeries Adapter .....	266

7.6.1. Introduction .....	266
7.6.2. MQSeries Installation .....	266
7.6.3. Typical Configuration and Message Conversion .....	266
7.6.4. Configuration .....	268
7.7. TIBCO Rendezvous Adapter .....	270
7.7.1. Introduction .....	270
7.7.2. Adapter Installation .....	270
7.7.3. Typical Configuration and Message Conversion .....	270
7.7.4. Configuration .....	271
7.7.5. Templates .....	273
7.8. Microsoft MQ Adapter .....	274
7.8.1. Introduction .....	274
7.8.2. Overview .....	274
7.8.3. MSMQ Installation .....	275
7.8.4. Configuration .....	275
7.8.5. Troubleshooting .....	276
7.9. JDBC Adapter .....	277
7.9.1. Overview .....	277
7.9.2. JDBC Adapter Installation .....	277
7.9.3. Configuration .....	277
7.9.4. Template .....	278
7.9.5. Repeating Groups Considerations .....	279
7.10. Flat File Adapter .....	279
7.10.1. Introduction .....	279
7.10.2. Operation .....	279
7.10.3. Configuration .....	280
7.10.4. File System Persistent Message Queue .....	280
7.10.5. File Naming Conventions .....	282
7.10.6. Queue Snapshots .....	283
7.10.7. Templates .....	284
7.11. RMI Adapter .....	284
7.11.1. Introduction .....	284
7.11.2. Configuration .....	284
7.11.3. Catalys Node Object Model .....	286
7.11.4. Class Summary .....	286
7.11.5. Templates .....	287
7.11.6. Troubleshooting .....	288
7.12. EMX Adapter .....	289
7.12.1. Introduction .....	289
7.12.2. Introducing the EMX Integrator .....	290
7.12.3. Installation .....	291
7.12.4. Running the Demonstration Programs .....	291
7.12.5. Developing Your System .....	297



7.12.6. Performing the EMX Compliance Tests .....	301
7.12.7. Obtaining a Certificate for the Live EMX System .....	303
7.12.8. Other Information .....	304
7.13. EMX Delimited File Adapter .....	304
7.13.1. Introduction .....	304
7.13.2. Configuration and Initial Set Up .....	305
7.13.3. EMX Delimited File Adapter User Guide .....	305
7.13.4. Daily Running .....	307
7.13.5. Outgoing Message Files .....	308
7.13.6. Incoming Message Files .....	311
7.13.7. Operational Issues .....	313
7.14. C/C++ API .....	314
7.14.1. Introduction .....	314
7.14.2. Overview .....	314
7.14.3. Programming Notes .....	315
7.14.4. Library .....	315
7.14.5. Class Diagram .....	316
7.14.6. Sequence Diagram .....	320
7.14.7. Templates .....	321
7.14.8. Sample user code .....	321
7.15. Microsoft .NET API .....	322
7.15.1. Introduction .....	322
7.15.2. Installation .....	323
7.15.3. Getting Started .....	323
7.15.4. The Catalys Node .NET API .....	323
7.15.5. Catalys Node Configuration .....	327
8. Programming Guide .....	330
8.1. Custom Components .....	331
8.1.1. Component Lifecycle .....	331
8.1.2. Application Lifecycle .....	334
8.1.3. Overriding the Lifecycle Methods .....	335
8.1.4. Custom Message Processors .....	339
8.1.5. Custom Selectors .....	344
8.1.6. Custom Services .....	346
8.1.7. Custom Monitors .....	348
8.1.8. Accessing Global Configuration and Runtime .....	350
8.1.9. JMX Component Operations .....	351
8.2. CLI Commands .....	354
8.2.1. Introduction .....	354
8.2.2. Implementation .....	354
8.3. Tasks .....	356
8.3.1. Introduction .....	356
8.3.2. TaskBase Implementation .....	356

8.3.3. PartyTaskBase Implementation .....	357
8.4. Persistent Objects .....	358
8.4.1. Introduction .....	358
8.4.2. Collection Registry .....	358
8.4.3. Compacting a Collection .....	360
8.4.4. Removing a Collection .....	360
8.4.5. Collections .....	360
8.4.6. Externalizers .....	362
8.5. Message Factories .....	364
8.5.1. Introduction .....	364
8.5.2. Implementation .....	364
8.6. Message Transformers .....	365
8.6.1. Introduction .....	365
8.6.2. Implementation .....	365
8.6.3. Configuration .....	366
8.7. Repeating Groups .....	366
8.7.1. Introduction .....	367
8.7.2. Adding a Repeating Group .....	367
8.7.3. Parsing Repeating Groups .....	367
8.8. JMX API Advanced Operations .....	368
8.8.1. Configuring via JMX .....	368
8.8.2. Monitoring via JMX .....	370
8.9. Authentication .....	371
8.9.1. Introduction .....	371
8.9.2. References .....	371
8.9.3. FIX Authentication Implementation .....	372
8.9.4. FIX Authentication Configuration .....	372
8.9.5. Login Module Configuration .....	372
8.9.6. Support for Custom Login Module Implementations .....	372
8.9.7. FIX Authentication Templates .....	373
9. Performance Tuning .....	374
9.1. Introduction .....	374
9.2. Measuring Performance .....	375
9.2.1. Measuring Throughput .....	375
9.2.2. Measuring Latency .....	376
9.2.3. Instrumenting Message Counts .....	377
9.3. Hardware Considerations .....	378
9.3.1. CPUs and Threading .....	378
9.3.2. Memory .....	378
9.3.3. Disk Storage .....	378
9.3.4. Network .....	378
9.4. Configuring for Performance .....	379
9.4.1. Threading .....	379

9.4.2. Logging .....	383
9.4.3. Message Validation .....	383
9.4.4. Persistence .....	383
9.4.5. Choosing an IFIXMessage Implementation .....	383
9.5. JVM Tuning .....	384
9.5.1. JVM Configuration .....	384
9.6. Custom Code .....	385
9.6.1. Message Object Reuse .....	385
9.6.2. Efficient Field Retrieval .....	386
9.6.3. Floating-point Types .....	387
A. Configuration Reference .....	388
B. CatalysServer Startup Arguments .....	699
C. FIX Session Dialects .....	701
C.1. Introduction .....	701
C.2. Configuring a Dialect .....	701
C.3. Available Dialects .....	702
C.4. Implementations Requiring Dialects .....	707
D. Persistent Collections Properties .....	711
D.1. Introduction .....	711
D.2. Properties .....	711
D.2.1. Common Collection Properties .....	711
D.2.2. Specific Collection Properties .....	715
D.3. Compacting Collections .....	720
E. Error Codes .....	722
E.1. Errors .....	722
E.1.1. General Errors .....	722
E.1.2. Send Errors .....	722
E.1.3. Bad Data Errors .....	722
E.1.4. Fatal Protocol Errors .....	723
E.1.5. Non Fatal Protocol Errors .....	725
E.1.6. Logon Errors .....	725
E.1.7. Message Errors .....	726
E.1.8. Routing Errors .....	728
E.1.9. Timeout Errors .....	728
E.1.10. SCP Errors .....	728
F. JMX MBean Reference .....	730
F.1. Platform MBeans .....	730
F.2. Server Management MBeans .....	732
F.2.1. Viewing Basic Node Information .....	732
F.3. Session Management MBeans .....	734
G. Third-Party Software .....	735

# Chapter 1. Introduction

## Table of Contents

1.1. Catalys Overview .....	1
1.2. Catalys Node Overview .....	2
1.3. Catalys Releases .....	3
1.4. What's New in Catalys 2.2 .....	4
1.4.1. Installers .....	4
1.4.2. New Features in Catalys Node 2.2 .....	4
1.4.3. Public API changes Catalys Node 2.2 .....	6
1.4.4. Known Issues in Catalys Node 2.2 .....	6
1.4.5. New Features in Catalys Node 2.1 Maintenance Releases .....	7
1.4.6. New Features in Catalys MIS 2.2 .....	10
1.4.7. Known Issues in Catalys MIS 2.2 .....	10
1.4.8. New Features in Catalys MIS 2.1 Maintenance Releases .....	12
1.4.9. New Features in Catalys LMA 2.2 .....	12
1.4.10. Known Issues in Catalys LMA 2.2 .....	12
1.5. Getting Support .....	13
1.5.1. Contacting Support .....	13
1.5.2. Sending Files to Your Support Team .....	14
1.6. FAQs .....	14

## 1.1. Catalys Overview

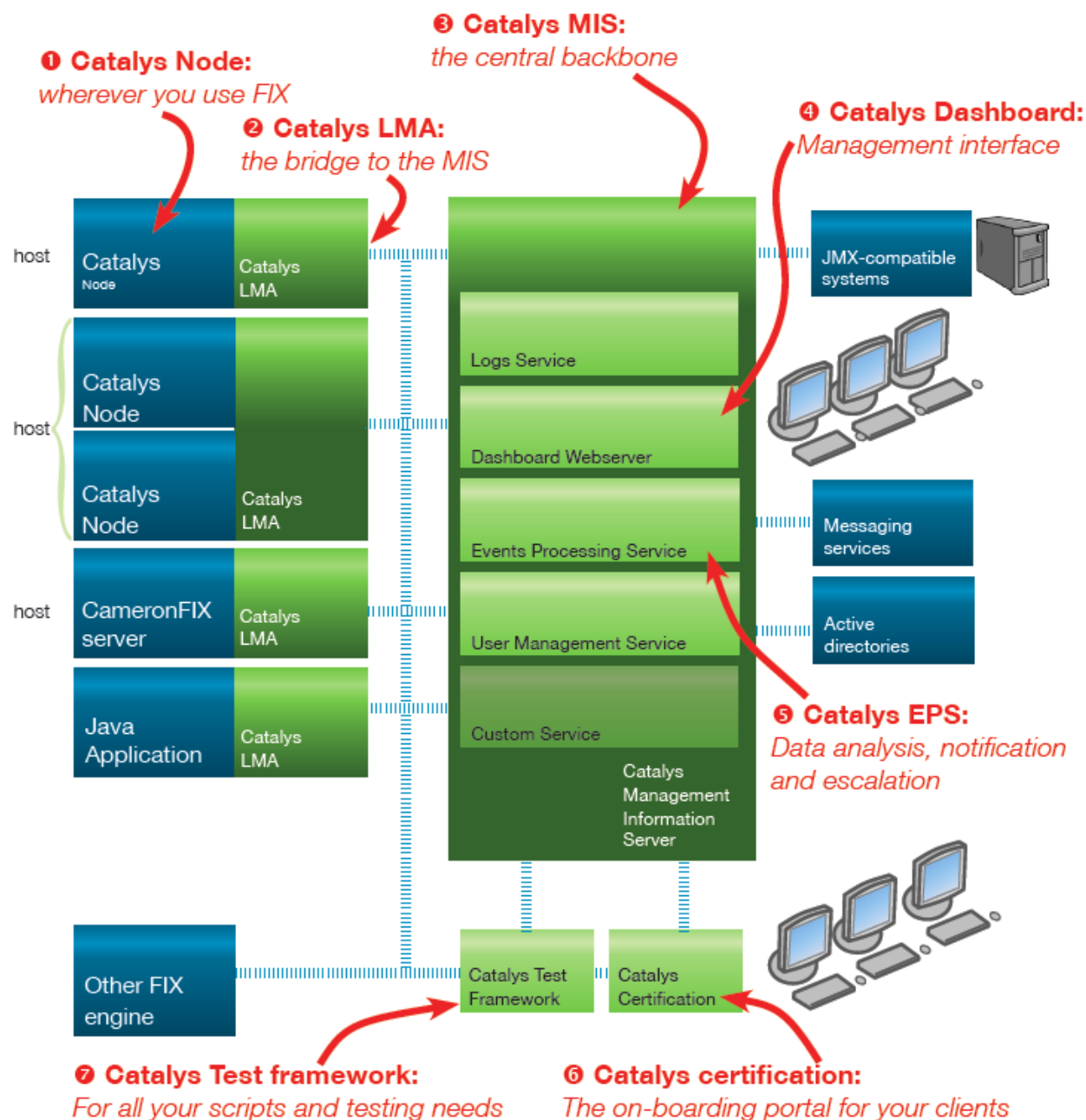
Catalys is [CameronTec](#)'s integrated software solution designed to meet the broad spectrum of functional needs for firms seeking to utilize advanced FIX connectivity.

What makes Catalys different is its unique approach: it is an unprecedented environment to develop, deploy and operate your advanced FIX infrastructure as an **ecosystem**.

Multiple components work together inside the Catalys ecosystem:

- The Catalys Nodes handle the FIX message flows. Refer to the Catalys Node Documentation.
- The Catalys LMA connects the nodes to the rest of the ecosystem. Refer to the Catalys LMA Documentation.
- The Catalys MIS is the central backbone for transversal services such as monitoring provided via an API or the Catalys Dashboard user interface. Refer to the Catalys MIS Documentation.
- The Catalys Test Framework produces and runs testing scripts for all aspects of the ecosystem. Refer to the Catalys Test Framework.

The various components are represented in the following diagram. For more information, please consult the specific section of the Catalys documentation.



## 1.2. Catalys Node Overview

The Catalys Node is an "out of the box" Java messaging application that supports the FIX protocol and provides communication with your internal systems.

The Node can simultaneously manage multiple FIX connections and multiple FIX versions. It can be configured in a number of different ways:

- **FIX Integration:** Standard technology adapters are used to pass FIX messages to non-FIX applications.
- **FIX Routing:** FIX sessions are setup to pass messages in one session and out another session. This configuration supports drop copy, session consolidation, a FIX routing hub and many other session-to-session configurations.
- **Embedded:** The Node is embedded in and runs inside the same process as another Java application. All features of the Node are supported.

The Node is highly configurable via a single [XML file](#), our web interface called Dashboard or the [JMX API](#). Runtime configuration changes are also supported using any of these methods.

The Node has an open architecture and a number of customization points. Our [Programming Guide](#) outlines the most common customizations.

## 1.3. Catalys Releases

### Introduction

Catalys products are delivered in three different types of releases: major, minor and maintenance.

### Major Releases

A major release delivers new functionality and occasionally includes changes to the public APIs, configuration schemas, data file formats and default configuration values which may not be backward-compatible. New features can also replace previous features that accomplish the same function.

An example of a new major release is 2.0

### Minor Releases

A minor release delivers new functionality and bug fixes and will be compatible with the previous release.

An example of a new minor release is 2.1

### Maintenance Releases

In between each major and minor release, CameronTec provides iterative updates called Maintenance Releases. These contain the baseline software delivered in the current release plus bug fixes and minor new features. Although it's not required, it is always recommended to download the latest maintenance release for evaluation as it may provide finer tuning, performance improvements or increased stability.

When a maintenance release is created, the product's version number does not change. Rather, the *revision* number is incremented. For example: 2.1 r37928.

## Release Notes

Each major release is accompanied by a Release Summary which provides a high-level view of the changes and new features.

Maintenance releases are accompanied by Release Notes that provide an incremental listing of the enhancements and resolved issues in each release.

Here is a sample Release Notes excerpt:

**r35564 released 2015-05-18**

ENHANCEMENTS	COMPONENTS	SUPPORT CASES
■ NODE-4752: Provide user feedback when scanning large journals	Buffering File Persistence	
RESOLVED ISSUES		
■ NODE-734: Re-enable the "s_reprocess" command	Core File Persistence	SERV-60346

All releases, documentation and release notes can be downloaded from the [Docs Portal](#).

## 1.4. What's New in Catalys 2.2

The following new features are available in Catalys 2.2 products at launch. In addition, Catalys 2.2 includes a large number of new features that were also back-ported to Catalys 2.1 as part of maintenance releases.

For any changes made after the initial release of Catalys 2.2, refer to the release notes for each maintenance release. Release notes are available from the [Docs Portal](#).

### 1.4.1. Installers

Catalys products are now distributed with an installer, automating the process of install, upgrade and uninstall. The installers are platform-specific and support Linux, Windows (32- and 64-bit) and most Unix-based operating systems.

### 1.4.2. New Features in Catalys Node 2.2

#### General

- NODE-4929: SendingTime can now be expressed with millisecond, microsecond, or nanosecond precision.
- NODE-4884: New `-version` command-line switch for Catalys Node.

- NODE-4910: `Session.maximumGapQueueSize` is no longer unbounded by default.
- NODE-3771: New consolidated drop-copy message processor for low impact drop-copy of another Catalys Node's message flow.
- NODE-4727: `MarketMirrorPopulator` now supports message object reuse.
- NODE-4745: Added JMX monitoring for multi-connection sessions.

## High Performance Logging

- NODE-4365: Improved performance with many threads logging under intensive load.
- NODE-2428: Display jar file details in stack traces.

## High Availability

- NODE-4596: Support for HA replication of a custom service's configuration.
- NODE-4674: HA data replication performance improvements. Introduced new multicast data replication option, as well as a new passive site replication feature allowing periodic pull-based data replication instead of continual push-based data replication. Also added a safety mechanism that will prevent a node from do a resynchronization if it is too far behind.

## Collections Framework

- NODE-4765: Collections can now be created at run-time instead of only at startup.
- NODE-4925: High-capacity lists now support set in the middle of the list instead of only at the end.
- NODE-4586: High-capacity lists can now share the same indexer thread.
- NODE-4817: Maps and high-capacity lists can now use an off-heap cache for fast, garbage-free caching of very large amounts of data.

## Buffering

- NODE-4350: Message buffers now support message object reuse.
- NODE-4826: Message buffer size is no longer excessively logged.
- NODE-4809: Serialization format changed to be faster and more compact.

## Solace Adapter

- NODE-5030: Provide configurable last resort action for consumed Solace messages which cannot be processed.
- NODE-4860: Add option to safeguard against message loss due to windowing.
- NODE-4856: Add option to clear CompIDs in consumed Solace messages before firing to FIX.



- NODE-4821: Create `SolaceEndpoint` component for routing Solace messages.

## Advanced Socket Adapter

- NODE-4862: New high-capacity message store option.

## Socket Adapter

- NODE-4841: Minimizing logging of undeliverable events when no socket client is connected.

## Scheduling

- NODE-4848: New option to execute missed reset tasks on startup.

## Multi-threading

- NODE-4617: More descriptive thread naming.
- NODE-4803: Enhanced thread pinning capabilities.

## Routing

- NODE-2286: `Link.ignoreSessionSide` attribute removed.

## 1.4.3. Public API changes Catalys Node 2.2

Catalys Node 2.2 is compatible with Catalys Node 2.1 except that the following XML configuration attributes are no longer necessary and have been removed:

- `ListenerMessageBuffer.setStoresMessages`
- `SourceMessageBuffer.setStoresMessages`
- `Link.ignoreSessionSide`

In addition there is currently a known issue causing an incompatibility between the persistence files of message buffers in 2.1 and 2.2. See issue NODE-5046.

## 1.4.4. Known Issues in Catalys Node 2.2

- NODE-5046: Message buffer persistence from 2.1 is not compatible with 2.2.
- NODE-5045: RulesEngine round() function is inconsistent with legacy valueOf() function.
- NODE-5038: HA synchronization window warning should only be logged once.
- NODE-5037: Unexpected exception while trying to reload config containing rules.

- NODE-5033: Spurious OpenHFT thread affinity GC warnings.
- NODE-5031: Exception when activating a component that is already active.
- NODE-5016: Installer does not accept relative or home directory-related paths.
- NODE-4999: Node installer reports incorrect required disk space.
- NODE-4978: Uninstall does not remove empty CameronTec directory.
- NODE-4977: Uninstall incorrectly reports that it will delete the parent directory of an installation.
- NODE-4871: MarketDataProcessor leaves a thread running after deactivation.
- NODE-4870: NotifyingMessageBuffer retry thread ignores interrupts.
- NODE-4790: .NET client intermittently reports error when disconnecting.
- NODE-4713: Catalys server should not log errors and warnings when shutdown.

## 1.4.5. New Features in Catalys Node 2.1 Maintenance Releases

The following features were added to Catalys Node 2.2 and also back-ported to Catalys Node 2.1 as part of maintenance releases, and have been available for some time.

### General

- NODE-4577: Safety feature: Duplicate instances of the same Catalys Node will no longer run at the same time.
- NODE-4379: Improved logging at startup.
- NODE-2834: Maven POM file now included in the distribution.
- NODE-4694: Support for explicitly specifying SSL protocols
- NODE-4351: Reload configuration from XML now disables affected sessions automatically.
- NODE-4682: New `license_reload` command.
- NODE-4360: New session configuration attribute `maximumAppQueueSize`.
- NODE-3176: New Multiplexed and non-blocking I/O capability for FIX sessions.
- NODE-3174: Move `SendingTime` generation off critical outbound latency path.
- NODE-3172: Add core affinity capability to `MessageAssembler` thread.
- NODE-230: Add ability to control the ordering of tags within an `IFIXMessage`.
- NODE-4558: Removed periodic object creation in `SessionManager`.

## High Availability

- NODE-4788: Ability to mark certain cluster nodes as not to be used as data sources during synchronization.

## Collections Framework

- NODE-4752: Progress is now logged periodically when scanning very large journals.
- NODE-4636: Collection directory uniqueness is now validated at startup.
- NODE-4747: Journaling persistent stores unnecessarily copy bytes when reading from journal

## Scheduling

- NODE-4878: EventLists are now validated so that two tasks will not execute at the same time.

## Buffering

- NODE-4763: Introduced output limit to `mb_messages` command.
- NODE-2212: New `mb_remove` command for removing messages from buffers.

## Embedded Adapter

- NODE-4380: Added ability to manage sessions from the embedded Catalys Node's owning application.

## IBM MQ Adapter

- NODE-4432: SSL support.

## Advanced Socket Adapter

- NODE-4525: New `asa_list` command that lists all connected ASA clients and the sessions they are subscribed to.
- NODE-4484: New CONNECT message that supports filtering by SessionID.
- NODE-4472: Added keepAlive configuration option.

## Solace Adapter

- NODE-4446: New Technology Adapter: Solace Adapter.
- NODE-4824: Support Solace JCSMP API v7.
- NODE-4823: Add support for reply-to topics.
- NODE-4822: Add support for dynamic error topics.

- NODE-4779: New Solace message transformer that ensures valid tag order, BodyLength and CheckSum.
- NODE-4571: Remove restriction on Solace message type.
- NODE-4567: DMQ support.

## .NET Adapter

- NODE-4767: Added ability to get session status from .NET application.

## Throttling

- NODE-4864: New inbound message throttling feature based on limiting the speed at which messages are read from the socket rather using message buffering.

## Routing

- NODE-2272: Any-to-any routing support. No need to define static session groups.
- NODE-4813: `AutoRoutingResponse` can now be placed anywhere in the message processing chain.

## Rules Engine

- NODE-4608: Added the concept of default rules.
- NODE-4619: New action: `ActionStripChars`.
- NODE-4734: New action: `ActionFileLookup`.
- NODE-4589: New action: `ActionLog`.
- NODE-4520: New condition: `ConditionFieldsUnique`.
- NODE-4547: New condition: `ConditionOvernightOrder` to hold and release outbound messages to FIX session based on a schedule.
- NODE-4611: `ActionReject` can now send application level rejects.
- NODE-4622: New function available in expressions: `round( )`.
- NODE-4497: Add the ability to pass the message processor object to custom actions.
- NODE-4496: Add ability to load rules from the config file and apply them to messages outside the Catalys FIX engine.
- NODE-4923: Add verbose output for rules engine tracking.
- NODE-2612: Rules Engine now supports message object reuse.
- NODE-4661: New option for handling Amend or Cancel messages where the original order cannot be found.
- NODE-4877: Reduce garbage creation in rules engine.

## 1.4.6. New Features in Catalys MIS 2.2

### General

- MIS-1563: LDAP connection timeout is now configurable.

### Logs

- NODE-2858: Introduced a new high-throughput logs streaming stack. Logs can now be transferred from Catalys Nodes to the MIS at approximately the bandwidth of the network connection.
- MIS-1677: New parallel log insertion algorithm increases the speed at which logs can be inserted into the log events repository.
- MIS-1584, MIS-1477: Log cleanup speed and memory usage drastically improved.
- MIS-1478: Browser memory usage stabilized when tailing logs for long periods of time.
- MIS-1593: Logs tab now provides visual feedback of ongoing searches.
- MIS-1598: Logs tab date selection widget is easier to use.
- MIS-1663: Logs cleanup configuration simplified.
- MIS-1646: New 'find' feature added to logs tab.
- MIS-1393: Custom conversation correlators can now define arbitrary custom message types.
- MIS-1392: Added ability to name custom tags.
- MIS-1699: Log streaming admin page redesigned.

### Events

- MIS-1661: New Action: Windows NT Event.
- MIS-916: Add Category filter to Log interests.

## 1.4.7. Known Issues in Catalys MIS 2.2

- MIS-1844: Logs streaming does not work on Solaris unless `LD_LIBRARY_PATH` contains `/usr/sfw/lib:/usr/sfw/lib/amd64`.
- MIS-1837: JMX cascading suffers poor performance over a slow network.
- MIS-1828: Logs cleanup is significantly slower when logs are being inserted at a high rate.
- MIS-1815: Replicated log files can become corrupt if source log files are modified in unexpected ways while being replicating.
- MIS-1812: Text Search - Escape button does not clear out search field.

## Introduction

- MIS-1811: Text Search - Search results should wrap around to start.
- MIS-1810: Events - Session Report interest taking union of selection when multiple interests configured.
- MIS-1805: Logs tab - Buttons are off screen while re-sizing left section panel.
- MIS-1796: MIS sometimes incorrectly reports that no logs were deleted during cleanup.
- MIS-1795: Admin - log streaming tab has double scroll bars and blank bottom section after exit from full screen mode.
- MIS-1790: Abrupt MIS termination can lead to duplicate log entries in the logs repository.
- MIS-1775: Events - Email password in mis.conf not encrypted.
- MIS-1773: Events - Escalation sequences - Stop-here-on-ack doesn't work correctly.
- MIS-1771: Admin - Permissions - Add escalation sequence permission only works if change escalation sequence permission is assigned.
- MIS-1770: Admin - Permissions - Add interest permission only works if change interest permission is assigned.
- MIS-1769: Admin - Permissions - Admin Tab is hidden unless incorrect set of permissions is assigned.
- MIS-1764: Servers and Sessions - Script info not shown for CameronFIX 8.0 servers.
- MIS-1763: Servers and Sessions - Process info not shown for CameronFIX 8.0 servers.
- MIS-1757: Config - Tooltips should be displayed for configuration tree icons.
- MIS-1756: Config - View changes so far menu item should be disabled when there are no changes.
- MIS-1755: Config - Add top-level selector should be disabled for 2.0 nodes.
- MIS-1754: Config - Unable to delete compression without multiple reloads.
- MIS-1753: Servers and Sessions - Change session sequence number doesn't work for 1.1 Catalys Nodes.
- MIS-1752: Events - Escalation Sequences - Event view of an escalation sequence's progress is faulty.
- MIS-1749: Events - Interests - Session report interest events are unreadable in dashboard.
- MIS-1747: Events - Interests - Log regex interest category field implies that it is a regular expression but is not.
- MIS-1745: Events - Add Interest - Underlined fields do not link to anything.
- MIS-1744: Events - Interest template description is not displayed.
- MIS-1743: Admin - Disabling a user while they are logged in causes their browser to repeatedly logout and login.
- MIS-1731: Uninstalling MIS on Windows does not warn if it cannot remove all directories.

- MIS-1709: Dashboard logged user out automatically after executing a log search after tailing logs overnight.
- MIS-1705: Multiple errors logged if replicated logs directory is deleted when logs replication and insertion into logs repository is in progress.
- MIS-1693: Logs tab apply button sometimes needs to be pressed twice.
- MIS-1640: When log filter end time is earlier than start time, an error comes up on dashboard.
- MIS-1610: Too many error message on dashboard if live streaming is on when shutting down MIS server.
- MIS-1581: If opening an interest after resizing the dashboard GUI smaller, the custom text fields are not popped out.
- MIS-1320: Events - Creation of an interest with incorrect regular expression is allowed by dashboard.
- MIS-461: Rules - Error when renaming rulesPack which is referenced before rules engine is reloaded.

## 1.4.8. New Features in Catalys MIS 2.1 Maintenance Releases

The following features were added to Catalys MIS 2.2 and also back-ported to Catalys MIS 2.1 as part of maintenance releases, and have been available for some time.

- MIS-1542: Log insertion can now be filtered by message type to exclude high-volume messages that do not need to be stored.
- MIS-1385: Added a calendar for public holidays to the interest schedules.
- MIS-1384: Session status column now takes session's schedule into account.
- MIS-1350: Open conversation view now retains user's current category selection.
- MIS-1301: Improved dashboard responsiveness when monitoring nodes with high latency.
- MIS-811: The MIS can now expose the output of scripts via JMX.
- MIS-1349: Conversation correlators added for SecurityDefinition conversations.

## 1.4.9. New Features in Catalys LMA 2.2

- NODE-2858: Introduced a new high-throughput logs streaming stack. Logs can now be transferred from Catalys Nodes to the MIS at approximately the bandwidth of the network connection.
- LMA-50: Unwrapped LMA distribution.

## 1.4.10. Known Issues in Catalys LMA 2.2

- LMA-65: Log4j guest logger support not handling `CR/LF` properly.

## 1.5. Getting Support

### 1.5.1. Contacting Support

#### Email

Send an email to your regional support team with the details of your issue or question and include log files and configuration files if applicable. This automatically creates a support ticket and alerts the support team.

- North America and South America: [supportamericas@itiviti.com](mailto:supportamericas@itiviti.com)
- Europe, the Middle East and Africa: [supportemea@itiviti.com](mailto:supportemea@itiviti.com)
- Asia Pacific: [supportapac@itiviti.com](mailto:supportapac@itiviti.com)

#### Online Form

[Submit a request](#) on our support portal. This automatically creates a support ticket and alerts the support team.

#### Phone

For high priority issues you can call your regional support team. This does not create a support ticket.

- North America and South America: **+1-312-235-5602**
- Europe, the Middle East and Africa: **+44-20-7942-0969**
- Asia Pacific: **+61-29002-4273**

#### Support Tickets

Emailing support or filling out the online support form automatically creates a support ticket, notifies the support team and sends the customer an automated email with a support ticket number. Any updates made to the ticket will be sent to the customer via email.

You can track and update all tickets submitted by your organization on the Support Portal by clicking on the [My activities](#) button on the top of page after logging in.

#### Support Level Agreement

During the License Term, as a part of the License Fee, Licensor undertakes to provide the software support set out in your contract, with respect to the Software Product. Such software support will be provided to the Customer through the Service Center during the Service Center's normal Business Hours: All Business Days between 8.00 a.m. – 6.00 p.m. local time for the respective Service Center.



The Service Center may choose to refer the calls to any other Licensor certified Service Center, or to the relevant third party Service Center as and when needed, during and outside its Business Hours.

## 1.5.2. Sending Files to Your Support Team

Most support issues require the support team to review certain configuration files or logs from your Catalys environment. These files will vary from product to product.

### Catalys Node

- **Configuration File(s):** One or XML files that contain the configuration for the Node instance. These are located in the `<CATALYS_NODE_INSTANCES>/<INSTANCE>/conf` directory if using the installer.
- **Log File(s):** One or more log files containing application and FIX message logging for the Node instance. These are located in the `<CATALYS_NODE_INSTANCES>/<INSTANCE>/logs` directory if you used the installer.
- **Log Configuration Files:** The *HighPerformanceLogger.conf* and *logback.xml* files for the Node instances. These are located in the `<CATALYS_NODE_INSTANCES>/<INSTANCE>/conf` directory if using the installer.
- **License File:** *camerontec\_license.xml* file located in the root of your CameronTec installation.

### Catalys LMA

- **LMA Configuration File:** *lma.conf* file located in the `/conf` directory of the standard LMA directory layout.
- **LMA Registry File:** *lma.registry* file located in the `/data` directory of the standard LMA directory layout.
- **LMA Log File:** The application log file for the LMA located in the `/logs` directory of the standard LMA directory layout.

### Catalys MIS

- **MIS Configuration File:** *mis.conf* file located in the `/conf` directory of the standard MIS directory layout.
- **MIS Log File:** The application log file for the MIS located in the `/logs` directory of the standard MIS directory layout.
- **License File:** *camerontec\_license.xml* file.

## 1.6. FAQs

[What are the key elements of the FIX protocol?](#)

[What is a "session"?](#)

[How does Catalys persist session data?](#)

[Why doesn't Catalys automatically persist inbound FIX messages?](#)

[What validation does Catalys perform on incoming messages?](#)

[Does my application need to handle FIX session-layer messages \(i.e. heartbeats and resend requests\)?](#)

[How can I monitor my Catalys Node instances?](#)

[How do I recover if my Catalys application crashes?](#)

[How can I change message sequence numbers on a session?](#)

[How can I customize the standard behavior of Catalys?](#)

[How does Catalys use threads?](#)

[Where can I find sample configurations?](#)

[Where can I find sample source code?](#)

[How do I work with repeating groups?](#)

[How can I improve the performance of my Node installation?](#)

[How do I write a FIX application using the Catalys API?](#)

## **What are the key elements of the FIX protocol?**

- There are multiple versions of the FIX protocol. Newer versions support additional message types, fields and values as well as deprecate or remove older message types, fields and values. The two parties of a FIX session agree on the FIX version that will be used.
- A FIX message consists of three parts: a header, a body and a trailer.
- Each part consists of a number of fields of the form tag=value separated by the special delimiter, ASCII "SOH".
- Apart from a few exceptions, fields can be in any order.
- Fields can be repeated, in which case the values are logically combined in some way.
- Fields have a type. For example, integer, float, character, time, date, binary data.
- All values except for binary data are encoded as ASCII strings.
- Binary data fields must be immediately preceded by a length field which is used to delimit the value. An important example of a binary data field is the "SecureData" field whose value contains one or more encrypted other fields. It is a field of encrypted fields.
- A message can be viewed as an unordered collection of fields.

- Each message has a type which defines a set of required fields and a set of optional fields.
- Each message contains a length and a checksum.
- Messages can be grouped into two categories: administrative and application.
- Administrative messages are used to support the protocol. Examples of such messages are: Heartbeat, ResendRequest, Logon and Logoff.
- Each message has a sequence number which increments in strict sequence, i.e. there should be no gaps. This is the basis of the protocol.
- A session is a logical connection (starting with a Logon and ending with a Logoff) comprising a continuous sequence of messages. A session can survive across multiple physical connections.
- A connection is a physical communications link between two parties.
- Fields can be encrypted.

## What is a "session"?

A FIX session is a socket connection between two parties. One party may be a professional trader and the other party may be a broker. A party is identified by a "CompID" and, optionally, a "SubID" and "LocationID".

## How does Catalys persist session data?

By default, Catalys persists the following data for all FIX sessions: inbound sequence number, outbound sequence number and all outbound messages. There are other features in Catalys that may also persist data, for example: [message buffering](#), the [Catalys Rules Engine](#) and custom code that utilizes our [persistent data objects](#).

In most cases it will be required to store the persistence data on disk, in case the application restarts for any reason. The built-in [JournalingPersister](#) provides this functionality.

Detailed information on the persistence feature can be found in [Session Persistence](#).

## Why doesn't Catalys automatically persist inbound FIX messages?

Catalys does not because it is unnecessary, inefficient and incorrect for the following reasons:

- The FIX protocol provides a mechanism to re-request messages from your counterparty for recovery purposes.
- Persisting inbound messages will add to inbound message latency.
- Most importantly, this is not correct. The FIX protocol does not require parties to do so and instead supports a ResendRequest message that requests the replay of message from one party to the other.

## What validation does Catalys perform on incoming messages?

See [Inbound Message Validation](#).

## Does my application need to handle FIX session-layer messages (i.e. heartbeats and resend requests)?

No. Catalys handles all the low-level FIX protocol messages for you automatically. Your application handles only the application message processing.

## How can I monitor my Catalys Node instances?

There are different options for monitoring the Catalys Node:

- Command Line Interface - the Node exposes a [command line](#) when running in "console" mode. You can also connect to running node using the [remote command line interface](#).
- Catalys MIS and Dashboard - the Management Information Server is an application that monitors Catalys Node applications. It provides a web interface called the Dashboard.
- [JMX API](#)
- Java API - by extending the [SessionStatusListenerBase](#) base class which exposes a number of event handlers within the Node. See the SampleMonitor template for more details.

## How do I recover if my Catalys application crashes?

Catalys should take care of most of the session recovery for you.

If your sessions are configured with file-based persistence then there should be no message loss. If after a restart a counterparty sends a ResendRequest, the Node will respond accordingly based on the FIX specification. If the Node detects that a session is missing messages, a ResendRequest will be sent to the FIX counterparty.

If your sessions cannot reconnect or are having sequence number issues then manual intervention may be required to get the session restarted.

## How can I change message sequence numbers on a session?

Caution must be taken when changing sequence numbers. There are catastrophic side effects to incorrectly changing sequence numbers.

Every FIX session has two associated sequence numbers: inbound and outbound. Sequence numbers must be in strict ascending order, and no gaps are allowed. These sequence numbers are the key to the FIX protocol. They are used to ensure that messages are processed in the correct order and that no messages are missing.

When two parties are communicating on a session, the input sequence number of one is the output sequence number of the other, and vice versa. If one party detects a gap in sequence numbers, it knows that message(s) have gone missing and can re-request them.

One of the cardinal rules of the FIX protocol is that one party cannot arbitrarily change sequence numbers without informing the other party.

If there is a sequence number conflict which cannot be resolved and both parties have agreed that no messages can be lost, the safest action is for both parties to disconnect and reset inbound and outbound sequence numbers to 1.

In rare instances (for example during testing) it may sometimes be convenient to set sequence numbers to some arbitrary value. This should only be done in extraordinary circumstances.

## How can I customize the standard behavior of Catalys?

There are numerous built-in application and session level settings and dialects that can change the default behavior.

Alternately you can create an [ISessionCustomizer](#) and register that object as a session listener. One convenient way of creating an [ISessionCustomizer](#) is to subclass [SessionCustomizerBase](#) and override the methods you want to change.

## How does Catalys use threads?

See [Threading](#).

## Where can I find sample configurations?

Templates can be found in the *templates/* subdirectory of your Node installation.

## Where can I find sample source code?

Source code examples can be found in the *templates/src/* subdirectory of your Node installation.

## How do I work with repeating groups?

As of Catalys 2.0, the repeating groups API is deprecated. See [IFIXMessage](#) for examples on how to add and iterate over repeating groups in a message object.

## How can I improve the performance of my Node installation?

See [Performance](#).

## How do I write a FIX application using the Catalys API?

See the example in *templates/CoreLibsSimple* and the accompanying *SimpleBuySide.java* and *SimpleSellSide.java* classes located in the *templates/src/com/camerontec/catalys/core/test* subdirectory of your Node installation.

# Chapter 2. Installation and Startup

## Table of Contents

2.1. Prerequisites .....	20
2.2. Support Portal .....	21
2.3. License Management .....	22
2.3.1. License File .....	22
2.3.2. How to Read Your License File .....	22
2.3.3. License Maintenance .....	23
2.4. New Installation .....	23
2.4.1. Introduction .....	23
2.4.2. Installation Steps .....	23
2.4.3. Post-Installation Core Node Directories .....	25
2.4.4. Post-Installation Node Instances Directories .....	25
2.4.5. Upgrading or Installing Additional Features .....	26
2.4.6. Uninstaller .....	26
2.5. Migrating a Previous Installation .....	27
2.5.1. Introduction .....	27
2.5.2. Java .....	27
2.5.3. License File .....	27
2.5.4. DTD Reference .....	27
2.5.5. Migrating Node Instances Without Using the Installer .....	28
2.5.6. Migrating Node Instances Using the Installer .....	28
2.5.7. XML Configuration .....	28
2.5.8. Logging Configuration .....	29
2.5.9. Persistent Data Files .....	29
2.5.10. High Performance Logger (HPL) Data Files .....	29
2.6. Startup .....	29
2.6.1. XML Configuration File .....	29
2.6.2. Startup Script .....	30
2.6.3. Starting the Example Node Instance .....	31
2.6.4. Troubleshooting .....	32

## 2.1. Prerequisites

### Java 1.8

The Catalys Node is a Java application built with Java 8. It can run on any operating system where a Java Runtime Environment (JRE) 8 or later is installed. No other third party libraries or databases are required.

## License File

Our License Team will provide you with an XML license file that contains a license key for the Node and any other products for which you are licensed. It will include an expiration date and all of your licensed features. See [License Management](#) for details on how to read and install the license.

## Write Access

Certain features of the Node (persistence, logging, rules engine, etc.) create and write to files on the file system. The user account under which the Node runs must have write access on these directories.

## 2.2. Support Portal

### Registration

To gain access to the CameronTec Group Support Portal you must request login credentials to your account manager or at Our support team will review and activate your account.

### Software Downloads

Installers for all Catalys products are available here on the [Docs Portal](#). After logging in, click on "Catalys / Cameron FIX" box and then click on the "Catalys FIX" link in the "Software Downloads and Documentation" box. From there you can select the version you require.

### Documentation

Documentation for all Catalys products is available from the [Docs Portal](#). After logging in, click on "Catalys / Cameron FIX" box and then click on the "Catalys FIX" link in the "Software Downloads and Documentation" box. From here you can view online versions of the documentation or download a PDF for offline reading.

### Release Notes

Each Catalys Node release is accompanied by up-to-date release notes that explain the incremental changes in the release. You can [download](#) the latest release notes from the same location as the software downloads.



## 2.3. License Management

### 2.3.1. License File

You will receive an XML license file with one or more license keys for the software you are permitted to run. To use the license file, the name of the file must be *camerontec\_license.xml*.

There are a number of ways to instruct the CameronTec software to find the license file. When a CameronTec application starts up it will search the following directories (in the order listed) until a license file is found:

1. Java system property specified in the startup command:

```
java -Dlicense-dir=/path/to/license/directory ...
```

2. CAMERON\_LICENSE\_ROOTDIR environment variable:

```
$ export CAMERON_LICENSE_ROOTDIR=/path/to/license/directory
```

3. Classpath of the Java application:

```
java -cp /path/to/license/directory:...
```

If you use an installer, you will have the option of importing a license file. If selected, the installer will prompt you for the location of the file, which will be copied into the root of the install and renamed accordingly. You must ensure that the license directory is included on the classpath of the Catalys application you are running.

### 2.3.2. How to Read Your License File

Your license is an XML file distributed by our license management team. One license file can contain multiple license keys for each product and version you are licensed for. It can be viewed with a text editor and is structured as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<camerontecLicense>
  <!-- Customer identifier -->
  <customer>Customer Name</customer>
  <!-- Licensed product. Multiple product keys can be present in one file. -->
  <product key="Catalys-Node-2.2">
    <license>
      <!-- Type of license, e.g. production or evaluation -->
      <licenseType>Evaluation</licenseType>
```

```

<!-- License expiration date at 23:59:59 -->
<expirationDate>yyyy-mm-dd</expirationDate>
<!-- Scope of authorised usage -->
<scope>Instance</scope>
<!-- Licensed product features -->
<features>
  <feature name="FAST" enabled="true" />
  <feature name="FIX Routing" enabled="true" />
  <feature name="High Availability" enabled="true" />
  <feature name="Scheduling" enabled="true" />
  <!-- Number of features (-1 is unlimited) -->
  <limitedFeature name="Sessions Count" count="-1" />
</features>
</license>
<!-- Product key signature, which cannot be modified -->
<signature>...</signature>
</product>
<product key="Catalys-MIS-2.2">...</product>
</camerontecLicense>

```

### 2.3.3. License Maintenance

License files expire periodically and customers may be required to replace their existing license. In most cases you can just overwrite the existing *camerontec\_license.xml* file with the new one.

## 2.4. New Installation

### 2.4.1. Introduction

The Catalys Node application has traditionally been packaged as a platform-independent zip file which is installed simply by unzipping the package in the desired destination directory. Starting with Catalys v2.2, we provide an automated platform-specific installer that separates the Node distribution from the Node instances. The installer also helps you to:

- Selectively install (and omit) certain features
- Automatically create our prescribed directory structure that eases management of your Node instances
- Simplify the upgrade process

We continue to provide the zip package for customers who prefer to manually manage their installation process. However, for all other customers, we recommend using the installer, which is covered in the remainder of this section.

### 2.4.2. Installation Steps

1. Download the installer for your platform:

- **Linux, Solaris, Mac OSX and other Unix-based systems:** *catalys-node-2.2-installer-r<revision>.sh*

After downloading this file, you may need to make it executable if not already. This can be done by running `chmod +x <filename>`.

- **Windows 64-bit:** *catalys-node-2.2-installer-64bit-r<revision>.exe*
- **Windows 32-bit:** *catalys-node-2.2-installer-32bit-r<revision>.exe*

The *r<revision>* indicates the latest [maintenance revision number](#).

Windows installers require a JRE to be installed that matches the architecture of the installer.

2. Launch the installer.
3. Select a Destination Directory: This is where the Node distribution will be installed.
4. Select Components:
  - a. **Core Node** - core binaries and files required to run the Node.
  - b. **Node Instances** - configuration, scripts and recommended directory layout for an instance of the Node.
  - c. **Configuration and Coding Templates** - XML configuration templates and sample Java classes developed against the Catalys Node API for the purposes of changing the default behavior, introducing business logic, adding services, monitoring and configuration.
  - d. **Maven Support** - includes a *pom.xml* file and unbundled *catalys-node.jar* to support building Java applications using Maven.
  - e. **C/C++ Adapter** - includes C/C++ libraries and example client code for connecting to the Node with a C/C++ application.
  - f. **MSMQ Adapter** - includes DLLs for connecting to the Node with the Microsoft Messaging Queue technology (only available in Windows installers).
  - g. **.NET Adapter** - includes DLLs and example client code for connecting to the Node with a .NET application (only available in Windows installers).
  - h. **Break Out Box** - includes a *web* directory that supports our Break Out Box functionality.
5. Select a Catalys Node Instances directory: Node Instances contains configuration files, scripts and data for individual Node instances. They are stored in a location separate from the core Node distribution in order to allow the core Node to be upgraded without affecting the instances. This also helps when migrating individual instances from one environment to the next (e.g. Dev -> UAT -> Production).



### Tip

If you select this option, files for an example Node instance will be included in the directory you choose. We recommend using this as a starting point for your own Node instances.

6. Install License File: Enable this step if you would like the installer to place a copy of your license file into the root of our installation. There are other ways to reference the license file which are outlined in [License Management](#).

## 2.4.3. Post-Installation Core Node Directories

After the installation is complete, you will see the following Core Node directories. The default location is `<INSTALL_BASE>/CameronTec/catalys-node-2.2`.

- **doc** - contains information on our 3rd party libraries and has links to our online documentation (included with "Core Node" option)
- **ext** - contains 3rd party libraries that cannot be packaged with *catalys-node.jar* due to licensing constraints (included with "Core Node" option)
- **lib** - contains *catalys-node.jar* and other optional library files based on the selected components (included with "Core Node" option)
- **lib-unbundled** - contains an unbundled *catalys-node.jar* and separate 3rd party jar files (included with the "Maven Support" option)
- **maven** - contains a Maven *pom.xml* (included with the "Maven Support" option)
- **resources** - contains default FIX dictionaries, Rules Engine validation files, DTD reference files and Rules Engine data files (included with "Core Node" option)
- **templates** - contains XML configuration examples showing how to configure different Catalys features (included with "Configuration and Coding Templates" option)
- **templates/src** - contains sample Java code written against the Catalys Node API (included with "Configuration and Coding Templates" option)
- **web** - contains the web GUI for Break Out Box feature (included with "Break Out Box" option)

## 2.4.4. Post-Installation Node Instances Directories

You will only see these directories if you selected the 'Nodes Instances' option when using the installer.

After the installation is complete, you will see the following Node instance directories. The default location is `<CATALYS_NODE_INSTANCES>/example`.

- **conf** - contains logging configuration and the configuration for this Node instance
- **data** - contains data directories and files created and used by this Node instance (e.g. FIX persistence, Rules Engine persistence, Buffering, Throttling)
- **logs** - contains ASCII-based log files for this Node instance
- **logs/hpl** - contains binary-based (HPL) log files for this Node instance
- **scripts** - contains scripts that work with this Node instance

The scripts that are included with the example Node instance are:

- **clean-data.[sh|bat]** - deletes all persistence data for the example instance
- **clean-logs.[sh|bat]** - deletes all log data for the example instance
- **start-cli.[sh|bat]** - starts the remote command line interface for the example instance
- **start-example.[sh|bat]** - starts the example Node instance process
- **view-hpl-logs.[sh|bat]** - outputs the HPL log data for the example instance

### 2.4.5. Upgrading or Installing Additional Features

The installer can be used for upgrading the Node to a newer version, starting at version 2.2 or later. It can also be rerun to add additional options that weren't selected on a previous installation. When the installer starts it should recognize that a previous installation exists.

The Node instance directory and its contents will never be updated or overwritten once created. Anything in the Node core directory may be overwritten or deleted by the installer.

### 2.4.6. Uninstaller

If you used the installer, an `uninstall` executable will be placed in the base directory of your Node. Run this to remove the entire Catalys Node distribution.



#### Note

Running the `uninstall` executable will not delete your `<CATALYS_NODE_INSTANCES>` directory or contents. You must manually delete this if required.

## 2.5. Migrating a Previous Installation

### 2.5.1. Introduction

This section will go over migrating from previous versions of Catalys. It will not cover migrating from previous versions of CameronFIX. Please see the PDF located [here](#) on our Support Portal for information regarding that.

### 2.5.2. Java

All Catalys 2.2 products are built using Java 8. You may be required to upgrade your Java to 8 or greater.

### 2.5.3. License File

You will need a new license file with license keys for the Catalys 2.2 products you are licensed to use. Please contact your account manager for a new license file with updated keys.

### 2.5.4. DTD Reference

The *CatalysConfig.dtd* file that all Node XML configuration files reference is now included in *catalys-node.jar*. You will need to update the previous relative references to this file in the */dtds* directory to point to the file name without any paths.

You need to change this:

```
<!DOCTYPE Config SYSTEM "../../../dtds/CatalysConfig.dtd">
```

To this:

```
<!DOCTYPE Config SYSTEM "CatalysConfig.dtd">
```



#### Note

It is still possible to support custom XML elements by adding them to your *CustomElements.dtd* file and using them in your Node configuration files. Please follow [these instructions](#) for help with migrating your custom elements.

## 2.5.5. Migrating Node Instances Without Using the Installer

If you installed the Catalys Node by downloading the unwrapped zip package, you will have all the directories listed in the [Post-Installation Core Node Directories](#) section. This layout is similar to previous versions of the Catalys Node. You will need to update the paths in your startup scripts to point to the new binaries as well as the references to *CatalysConfig.dtd*, as covered in the previous [DTD Reference](#) section.



### Note

It is recommended that you separate your Node instance configuration files, startup scripts, logs and persistent data in a directory outside of the *catalys-node-2.2* distribution.

## 2.5.6. Migrating Node Instances Using the Installer

If you installed the Catalys Node by using the installer, you will have some or all of the directories listed in the [Post-Installation Core Node Directories](#) section depending on the options you selected. You will need to update the paths in your startup scripts to point to the new binaries as well as the references to *CatalysConfig.dtd*, as covered in the previous [DTD Reference](#) section.



### Note

It is recommended that you separate your Node instance configuration files, startup scripts, logs and persistent data in a directory outside of the *catalys-node-2.2* distribution. If you selected the 'Node Instances' option, an */example* instance will be created. We recommend following the directory layout used here.

## 2.5.7. XML Configuration

Other than the [DTD update](#) mentioned above, XML configuration files from Catalys Node 2.0 and 2.1 require no modification after upgrading to Node 2.2. However, new elements and attributes have been added to support the new functionality. Please review the [What's New](#) document for a list of additions.

## 2.5.8. Logging Configuration

The Logback and HPL logging configuration files used in Catalys Node 2.0 and 2.1 require no modification after upgrading to Node 2.2.

## 2.5.9. Persistent Data Files

Persistent data files created by Catalys Node 2.1, other than data files created by the [Buffering](#) and [Throttling](#) components, require no modification or conversion after upgrading to Node 2.2.



### Warning

Data files created by the **Buffering** and **Throttling** components from previous versions of the Node must be deleted in order for the Node 2.2 to startup.

## 2.5.10. High Performance Logger (HPL) Data Files

HPL data files created by the Catalys Node 2.1 require no modification or conversion after upgrading to Node 2.2. Node 2.2 will append to the end of the 2.1 HPL files.

## 2.6. Startup

To start an instance of the Node, an XML configuration file, a valid license and a startup script are necessary.

### 2.6.1. XML Configuration File

The entire configuration for a Node instance is contained in an XML file. The following shows the XML for the example Node instance:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Config SYSTEM "CatalysConfig.dtd">
<Config>
  <Application id="example">
    <Services>
      <!-- Service that connects to the LMA & MIS/Dashboard for monitoring and exposes
           the remote command line interface via JMX, the host:port combination must
           be unique for each instance -->
      <JMXManagementService
        id="JMXManagementService"
        jmxServiceURL="service:jmx:jmxmp://localhost:1500"
        alwaysKeepJMXConnectorServerRunning="true" />
```



```

</Services>
<Sessions>
  <!-- Client/buy-side/initiator FIX session -->
  <Session counterpartycompid="BROKER" compid="BUY1" fixversion="4.4" heartbeat="60">
    <Persister>
      <!-- File based persistence -->
      <JournalingPersister id="jpl" baseDirectoryName="../data/persistence/broker"/>
    </Persister>
    <Connections>
      <!-- Session configured as a initiator on the session -->
      <SocketConnection id="sc1" hostname="localhost" port="2002"/>
    </Connections>
    <SessionManager>
      <SourceMessageProcessors>
        <!-- Built-in utility that sends an outbound test
              NewOrderSingle message every 15 seconds -->
        <SampleMessageSource id="sms1" checkBeforeSend="true" interval="15000" />
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
</Sessions>
</Application>
</Config>

```

The Node's configuration is comprehensively covered in the [following chapter](#).

### 2.6.2. Startup Script

The Node's startup script is a Unix shell script or a Windows batch file. It typically contains the following key elements:

- Java classpath including *catalys-node.jar* (typically in the *lib* directory of the installation) and the directory containing the license file and logging configuration.
- Java command referencing the startup class for the Node:  
`com.camerontec.catalys.server.CatalysServer`
- Application arguments, including the requisite `-xmlconfig <config-file>` argument. For a complete list, please see the [CatalysServer Startup Arguments](#) appendix.

A very simple startup script with these elements would look like:

```

java -cp "../lib/catalys-node.jar:../resources" com.camerontec.catalys.server.CatalysServer \
  -xmlconfig config_example.xml

```

The Unix-based startup script included with the example Node instance is here:

```

#!/bin/sh

```

## Installation and Startup

```
# if your path doesn't contain a JVM add it here
# export PATH=/path/to/jdk/bin:$PATH

# since the paths below are relative we need to make sure we are located
# in the /scripts directory
cd `dirname $0`

# change these values if using a different Node location
license_home=/Applications/CameronTec
node_home=/Applications/CameronTec/catalys-node-2.2

# change these values if using a different instance dir layout or config file
instance_home=.
instance_config=${instance_home}/conf/config_example.xml

# set up the classpath
cp=${instance_home}/conf
cp=${cp}:${node_home}/lib/catalys-node.jar
cp=${cp}:${node_home}/ext/*
cp=${cp}:${node_home}/resources
cp=${cp}:${license_home}

java -Xmx256M -classpath "$cp" \
    -DJMXManagementService.startupScript="$(cd $(dirname $0);pwd)/$(basename $0)" \
    com.camerontec.catalys.server.CatalysServer -xmlconfig ${instance_config}
```



### Note

If a Node instance or the license file is moved to a different directory, you will need to update the `license_home`, `node_home` and `instance_home` script variables.

## 2.6.3. Starting the Example Node Instance

If you select the "Node Instance" option from the installer, files for an example Node instance will be included in the location you choose.

The `<CATALYS_NODE_INSTANCES>/example/conf/config_example.xml` configuration file defines an instance of the Node called "example" and contains one FIX session with the minimum required configuration elements. The FIX session initiates a FIX connection on port 2002 of the localhost and sends a Logon message after establishing a successful connection. After the Logon exchange, it will send a NewOrderSingle message every 15 seconds.

To start the example instance call the `<CATALYS_NODE_INSTANCES>/example/bin/start-example.[bat/sh]` script.

You will see logging on the console of the example as well as log files in the `log` directory of the installation.

If you are experiencing problems please use the next section for solving common startup issues. If you are still having problems, then contact our [Support Team](#).

### 2.6.4. Troubleshooting

Below are commonly encountered startup issues and their resolutions.

- Java not installed

```
java: command not found
```

A Java Runtime Environment (JRE) version 8 or higher must be installed to run the Node.

- Java version mismatch

```
Exception in thread "main" java.lang.UnsupportedClassVersionError: \
com/camerontec/catalys/server/CatalysServer: Unsupported major.minor version 51.0
```

A Java Runtime Environment (JRE) version 8 or higher must be installed to run the Node.

- CatalysServer not on classpath

```
Error: Could not find or load main class com.camerontec.catalys.server.CatalysServer
```

If you are using the installed startup script, check that the `node_home` script variable is set correctly.

If you are using your own startup script, check that the path to the *catalys-node.jar* is set correctly on the classpath.

- License not found

```
License Error: Could not find 'camerontec_license.xml' in classpath
```

If you are using the installed startup script, check that the `license_home` script variable is set correctly.

If you are using your own startup script, check that the directory containing *camerontec\_license.xml* is set correctly on the classpath.

More information on licensing is located [here](#).

- License version mismatch

```
License Error: Product "Catalys-Node-2.2" is not present in the license
```

You are attempting to run a version of the Node that does not match the version in your license. Please contact your account manager for an updated license.

## Installation and Startup

- Duplicate instance detected

```
ERROR [main] com.camerontec.catalys.server.CatalysServer - \  
    There is already an application listening on /127.0.0.1:12277 ...
```

This likely indicates that you have started the same Node instance twice. Measures are in place to prevent this as it could cause critical data to be overwritten. Please refer to the [instanceDetectionPort](#) attribute on the `Application` element for more details.

- Unlicensed features

```
License Error: Feature 'Socket Adapter' is not enabled in the license
```

You are attempting to use a feature of the Node that you are not licensed for. Please contact your account manager if you would like to evaluate this feature.

- Missing write permissions

```
ERROR in ch.qos.logback.core.rolling.RollingFileAppender[SERVERLOG] - \  
    openFile(catalys.log,true) call failed. \  
    java.io.FileNotFoundException: catalys.log (Permission denied)
```

The user account running the Node does not have write permissions to the *log* and/or *data* directories. Please grant write permissions to this user account.

- TCP port in use

```
ERROR [ServerConnectionManager-5001] CameronTec.ServerConnectionManager.5001 ... \  
    java.net.BindException: Address already in use
```

The port configured for the FIX session's `SocketConnection listenport` is already in use by another application on the server. Choose an alternate port.

# Chapter 3. Configuration

## Table of Contents

3.1. Overview .....	34
3.1.1. Application Configuration .....	35
3.1.2. Services Configuration .....	35
3.1.3. Session Configuration .....	36
3.1.4. Connection Configuration .....	37
3.1.5. Persister Configuration .....	37
3.1.6. Session Manager Configuration .....	38
3.1.7. Instance and Session Properties Configuration .....	42
3.1.8. Message Processing Templates Configuration .....	42
3.1.9. Replacement Parameters .....	43
3.2. Logging .....	44
3.2.1. Choosing a Logging Implementation .....	44
3.2.2. HPL Configuration and Operation .....	45
3.2.3. Logback Configuration .....	48
3.2.4. FIX Message Logging .....	51
3.2.5. Logging API .....	52
3.3. JMX Configuration API .....	53
3.3.1. Introduction .....	53
3.3.2. Management Infrastructure .....	53
3.3.3. Configuration versus Runtime .....	54
3.3.4. Reload .....	54
3.3.5. Refresh .....	54
3.3.6. Undo .....	55
3.3.7. Persistence .....	55
3.3.8. MBeans .....	55
3.3.9. Interactively Exploring the API .....	56
3.3.10. Example .....	57
3.4. Encryption of Attributes .....	57
3.4.1. Encrypt an Attribute .....	58
3.4.2. Decrypt an Attribute .....	58

## 3.1. Overview

The entire configuration for a Catalys Node instance is contained in an XML file. It has the following overall structure:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE Config SYSTEM "CatalysConfig.dtd">
<Config>
  <Application>
    <Services>
      <!-- instance-level services -->
    </Services>
    <Sessions>
      <Session counterpartycompid="CLIENT1" compid="BROKER">
        <!-- session-level configuration -->
      </Session>
      <Session counterpartycompid="CLIENT2" compid="BROKER">
        <!-- session-level configuration -->
      </Session>
    </Sessions>
  </Application>
</Config>
```

The following sections cover the XML configuration for the basic features that most Node instances will use. Configuration and information for other [standard](#) and [optional](#) features are covered in later chapters.

### 3.1.1. Application Configuration

Only one `Application` element is allowed per configuration file. It represents a single Node instance. The value of the `id` attribute determines how an instance is known to other Catalys applications (specifically the LMA). This attribute is not required and defaults to `default`.

The other `Application` attributes control the behavior of the instance. Most changes to these values require a restart of the instance.

For a complete list of attributes, see the [Application Configuration Reference](#).

### 3.1.2. Services Configuration

The `Services` section contains instance-level service configuration. Normally services are objects whose functionality is shared by one or more FIX sessions. They can also provide instance-level functionality unrelated to FIX sessions. Services are named and exist in a global namespace within the Node. The most frequently used service is the [JMXManagementService](#).

For information on developing and configuring a custom service, see the [Programming Guide](#).

## JMXManagementService

There are a number of built-in services in the Node. The most frequently used one is the `JMXManagementService`. This service automatically connects the Node to the LMA, which allows for remote management. It can also expose the [Remote Command Line](#) functionality.

The `JMXManagementService` is configured and enabled simply with this configuration:

```
<Services>
```

```
<JMXManagementService id="JMXManagementService"/>
</Services>
```

For a complete list of attributes, see the [JMXManagementService Configuration Reference](#).

### 3.1.3. Session Configuration

Communication between two parties takes place on a session running a particular version of the FIX protocol over a socket connection. For the purposes of configuration, each session is defined by a counterPartyCompID/compID pair. This pair, plus the FIX version, are the minimum required session attributes:

```
<Session counterpartycompid="BROKER" compid="EXCHANGE" fixversion="4.2">
```

Sessions configured for FIX versions 4.0 to 4.4 require only the `fixversion` attribute to be set. For versions 5.0 and greater, the `transportVersion` attribute must also be configured:

```
<Session counterpartycompid="BROKER" compid="EXCHANGE" fixversion="5.0SP1" transportVersion="1.1">
```

The following is a complete, although simple, session configuration:

```
<Sessions>
  <Session counterpartycompid="BROKER" compid="EXCHANGE" fixversion="4.4" heartbeat="60">
    <Connections>
      <!-- Server-side session listening on port 5001 -->
      <SocketConnection id="scl" listenport="5001" />
    </Connections>
    <Persister>
      <!-- Default file-based persister -->
      <JournalingPersister id="jpl" baseDirectoryName="./data/persist/broker"/>
    </Persister>
    <SessionManager>
      <SourceMessageProcessors>
        <!-- Outbound message processing chain -->
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <!-- Inbound message processing chain -->
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
  <!-- ... Repeated for each session ... -->
</Sessions>
```

Each Session can also have its own [Scheduling](#), [Dialects](#), [Persistence](#), [Session Manager](#) and [Encryption](#) settings. A Session may also contain a [Properties](#) element which allows you to associate any other information you wish with the session.

For a complete list of Session attributes, see the [Session Configuration Reference](#).

### 3.1.4. Connection Configuration

The communication for a FIX session takes place over a socket connection. One party is a "listener", or server, listening on a specific port; the other party is an "initiator", or client, that connects to the server's IP address and TCP port. After the socket connection is established a Logon exchange takes place.

#### Server-Side Connections

Server-side or listening connections listen for a counterparty to connect to them. Multiple FIX sessions on a Node instance can listen on the same port so that only one inbound port needs to be open on a firewall.

A server-side connection is configured as follows:

```
<Connections>
  <!-- FIX session listening on port 5001 -->
  <SocketConnection id="sc1" listenport="5001" />
</Connections>
```

#### Client-Side Connections

Client-side or initiating connections attempt to establish a socket connection with a counterparty that is listening on a port. There may be one or more socket connections configured on client-side sessions. The Node will attempt all configured connections, in the order they are listed, until successful. This allows you to configure a primary connection and one or more backup connections, should your counterparty provide them.

The default connection behavior is to attempt to connect to the configured socket connections in the order they are listed in round robin order. Retry attempts are unlimited with a 10-second delay between each. This behavior can be changed in the [Session Manager](#) configuration.

Client-side connections are configured as follows:

```
<Connections>
  <!-- FIX session attempting to connect to counterparty at primary hostname and port -->
  <SocketConnection id="sc1" hostname="primary.server.com" port="5001" />
  <!-- FIX session attempting to connect to counterparty at secondary hostname and port -->
  <SocketConnection id="sc2" hostname="secondary.server.com" port="5001" />
</Connections>
```

[SSL Socket Connections](#) are also supported.

For a complete list of attributes, see the [Connection Configuration Reference](#).

### 3.1.5. Persister Configuration

See [Session Persistence](#) for information on this feature.



### 3.1.6. Session Manager Configuration

The Session Manager oversees several aspects of a Session:

- Socket connection and retry behavior.
- Dispatching outbound messages through the source message processing chain.
- Dispatching inbound messages through the listener message processing chain.
- Connections to non-FIX applications via our technology adapters.
- References to [message processing templates](#).

The Session Manager is always present even if the session is not connected and each session has exactly one Session Manager.

For a complete list of attributes, see the [SessionManager Configuration Reference](#).

### Message Processing Chain Configuration

The message processing chains (source and listener) of a session define the path of processing that each message takes. A message processing chain consists of message processors, technology adapters and selectors. These can be chained together and/or nested in such a way that certain messages follow one path and others follow an alternative, or it can be as simple as a single message processor that handles all messages.



#### Important

There are two independent message processing chains for each FIX session:

- **Source:** The path that outbound FIX messages take from the application to the FIX session.
- **Listener:** The path that inbound FIX messages take from the FIX session to the application.

### Message Processors

A message processor can be a source, a listener or both. To reiterate, a source message processor handles outbound FIX messages and a listener message processor handles inbound FIX messages. A message processor can be configured on both sides as well. For instance, an inbound message can trigger an outbound reject message back to the counterparty.

There are three main groups of message processors:

- **Technology Adapters:** Message processors that pass messages to/from a non-FIX application. We support a number of different adapters out of the box: [Solace](#), [IBM MQ Series](#), [Embedded](#), [JMS](#), [Tibco](#), [MSMQ](#), [Socket](#) (for Java, C/C++ and .NET clients), [RMI](#), [JDBC](#) and [Flat File](#). Note that technology adapters need to be the innermost nested message processors in the chain.
- **Built-in Message Processors:** Many built-in features of the Node are actually message processors: [Rules Engine](#), [Buffering](#), [Throttling](#) and [Scheduled Order Cache](#), for example.
- **Custom Message Processors:** The Node allows users to change and customize the standard behavior. The most common customization is in the form of custom message processors. See the [Programming Guide](#) for more information.

## MessageProcessorReference Configuration

Message processors that reside on both the source and listener chain can be defined as separate instances or as a single instance that is referenced from the other chain. The [MessageProcessorReference](#) element is used to refer to a previously defined message processor. In the following example, a single instance of the `CatalysRulesEngine` is defined on both chains. This is useful if inbound and outbound messages must pass through the same set of rules.

```
<SessionManager>
  <SourceMessageProcessors>
    <!-- CatalysRulesEngine initially defined here -->
    <CatalysRulesEngine id="crel" rulesPackId="store-and-restore-symbol"/>
  </SourceMessageProcessors>
  <ListenerMessageProcessors>
    <!-- MessageProcessorReference used to reference crel -->
    <MessageProcessorReference id="ref" refid="crel"/>
  </ListenerMessageProcessors>
</SessionManager>
```

A message processor defined in one session may also be referenced from another session:

```
<Sessions>
  <Session counterpartycompid="CLIENT1" compid="BROKER" ...>
    <SessionManager>
      <SourceMessageProcessors/>
      <ListenerMessageProcessors>
        <!-- GenericProcessor initially defined here -->
        <GenericProcessor id="gpl" class="com.camerontec.catalys.InboundStats"/>
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
  <Session counterpartycompid="CLIENT2" compid="BROKER" ...>
    <SessionManager>
      <SourceMessageProcessors/>
      <ListenerMessageProcessors>
        <!-- MessageProcessorReference used to reference gpl -->
        <MessageProcessorReference id="ref" refid="gpl"/>
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
</Sessions>
```

```

    </SessionManager>
  </Session>
</Sessions>

```

## Message Processor Nesting

If more than one message processor is required in a message processing chain, then the nesting order needs to be correctly configured.

For the source (outbound) message processing chain, messages will pass from the innermost element to the outermost element:

```

<SessionManager>
  <SourceMessageProcessors>
    <!-- Outermost element fires last and throttles outbound messages at 100 / sec -->
    <SourceMessageThrottle id="sml1" messagesPerInterval="100" directory="./data/throttle">
      <!-- Messages are passed to the CatalysRulesEngine from the JMSConsumer for enrichment -->
      <CatalysRulesEngine id="cre1" rulesPackId="outbound-enrichment-rules">
        <!-- Innermost element fires first and consumes message off a JMS queue -->
        <JmsConsumer id="jms1" type="queue" destination="BUY_OUTBOUND" factory="primaryQCF"/>
      </CatalysRulesEngine>
    </SourceMessageThrottle>
  </SourceMessageProcessors>
  <ListenerMessageProcessors/>
</SessionManager>

```

For the listener (inbound) message processing chain, messages will pass from the outermost element to the innermost element:

```

<SessionManager>
  <ListenerMessageProcessors>
    <!-- Outermost element fires first -->
    <GenericProcessor id="gpl" class="com.camerontec.catalys.CustomMP">
      <!-- Messages are passed here next - it buffers if the JMSProducer is down -->
      <SourceMessageBuffer id="smb1" directoryName="./data/buffer/smb1">
        <!-- Innermost element fires last and publishes message to a JMS queue -->
        <JmsProducer type="queue" destination="BUY_INBOUND" factory="primaryQCF" id="jms1"/>
      </SourceMessageBuffer>
    </GenericProcessor>
  </ListenerMessageProcessors>
  <SourceMessageProcessors/>
</SessionManager>

```

## Selectors

Selectors are used to control which message processors a message passes through. The most common use case is to control the processing of inbound messages by message type or some other FIX field criteria. Selectors are configured in the listener message processor chain of a session; they are not supported on the source chain. If a message processor is nested beneath a selector, then only messages

## Configuration

that have been "selected" will be passed to that message processor. Messages that are not selected will be passed on to any other message processors in the chain.



### Note

If any message processors are configured in the listener chain, then a message must pass through at least one of them. An exception will occur otherwise.

The Node has two built-in selectors: [SelectByTagValue](#) and [MsgTypeSelector](#). The source code for both classes is included in the templates. An example using `SelectByTagValue`:

```
<SessionManager>
  <ListenerMessageProcessors>
    <!-- Select certain messages by msgType and symbol and place on an inbound solace topic -->
    <SelectByTagValue selection="35=D 55=anyOf(GOOG,IBM)" id="sbtv1">
      <SolaceMessageListener id="solaceListener1" destinationTopic="GOOG_IBM_Orders"/>
    </SelectByTagValue>
    <!-- All remaining messages go to default solace topic -->
    <SolaceMessageListener id="solaceListener2" destinationTopic="topicAllOthers"/>
  </ListenerMessageProcessors>
  <SourceMessageProcessors/>
</SessionManager>
```

For a complete list of attributes, see the [SelectByTagValue Configuration Reference](#).

An example using `MsgTypeSelector`:

```
<SessionManager>
  <ListenerMessageProcessors>
    <!-- Select certain messages by msgType place on an inbound solace topic -->
    <MsgTypeSelector includeMsgTypes="D F G" id="mts1">
      <SolaceMessageListener id="solaceListener1" destinationTopic="AllOrderMessages"/>
    </MsgTypeSelector>
    <!-- All remaining messages go to default solace topic -->
    <SolaceMessageListener id="solaceListener2" destinationTopic="AllOthers"/>
  </ListenerMessageProcessors>
  <SourceMessageProcessors/>
</SessionManager>
```

For a complete list of attributes, see the [MsgTypeSelector Configuration Reference](#).

For information on developing your own selector, see the [Programming Guide](#).

### 3.1.7. Instance and Session Properties Configuration

Most configuration elements have named attributes such as `compid="BROKER"`. However, sometimes it is useful to associate an arbitrary, variable number of attributes with an element. In that case, a `Properties` element can be helpful.

For example, the `Session` element can contain a `Properties` element which stores arbitrary information about that session (i.e. Counterparty Contact Name and Phone Number). The attributes specified in a `Properties` element are added to the standard attributes specified in the parent element.

A `Properties` element contains a number of `Property` elements:

```
<Properties id="prop1">
  <Property name="..." value="..." />
</Properties>
```

`Properties` elements have the following characteristics:

- A collection of name/value pairs.
- Translates directly to a standard Java [Properties](#) object.
- Must be given an `id` attribute which is unique and can be used to refer to this element from other elements.
- Can be present anywhere in the configuration file.
- If you are using the MIS/Dashboard to monitor a Node instance and the `name` of a `Property` element begins with #, the name/value pair will be present in the Instance and Sessions grid.
- If another configuration element contains a `Properties` element, its values are added to the standard properties of the element. If an attribute of the same name appears in both the standard attributes and in the `Properties`, the value of the latter is used.

### 3.1.8. Message Processing Templates Configuration

[MessageProcessingTemplate](#) elements are used to define common chains of message processors which can be reused across multiple sessions. When several FIX sessions in one instance use the exact same message processing chain, message processing templates can ease configuration maintenance and significantly reduce the size of the configuration file.

First define a `MessageProcessingTemplate`:

```
<Application>
  <MessageProcessingTemplate id="mpTemplate1">
    <SourceMessageProcessors>
      <SampleMessageSource interval="${intervalVal}" id="sms_${session}" />
    </SourceMessageProcessors>
    <ListenerMessageProcessors>
```

## Configuration

```
<SampleMessageListener name="Received on ${session}" id="sml_${session}" />
</ListenerMessageProcessors>
</MessageProcessingTemplate>
<Sessions>
    ...
</Sessions>
</Application>
```

Note that attribute values can contain parameters using the `${parameter_name}` syntax. This allows those attributes to be replaced with session-specific values. The `${session}` parameter is [built-in](#).

Then, from each session, use a [MessageProcessingInstance](#) to apply the template:

```
<Session>
  <SessionManager>
    <MessageProcessingInstance refid="mpTemplate1">
      <Properties id="properties1">
        <Property name="intervalVal" value="10000" />
      </Properties>
    </MessageProcessingInstance>
  </SessionManager>
</Session>
```

See the template located in `templates/MessageProcessingTemplates` for a working example.

### 3.1.9. Replacement Parameters

Catalys supports the use of *replacement parameters* throughout the configuration file. These can be used to dynamically refer to environment variables and system properties. The replacement parameter `${env:envVar}` will be replaced with the value of the environment variable `envVar`. Similarly the replacement parameter `${system:sysProp}` will be replaced with the value of the system property `sysProp`.

Note, that *replacement parameters* can't be used in `id` attribute and attributes that refer to other elements using their `id`.

The following example illustrates how to configure the persister `baseDirectoryName` attribute using an environment variable:

```
<Session counterpartycompid="CTRCOMPID" compid="COMPID" fixversion="4.2" heartbeat="60">
  <Persister>
    <JournalingPersister id="jpl" baseDirectoryName="${env:envDir}/persist"/>
  </Persister>
  ...
</Session>
```

This will create persistence files whose names start with *persist-* in the directory referenced by the `envDir` environment variable.



### Note

Replacement parameters can only be used in values of attributes of type `String`. Please refer to the respective elements in the [Configuration Reference](#) for details on each attribute data type.

## Session Replacement Parameter

A built-in replacement parameter, `${session}`, can be used in message processing templates and as an attribute value within any `Session` element. The parameter will be replaced with the session's ID: `applicationID~counterpartyCompID~compID`. For example:

```
<Session counterpartycompid="CTRCOMPID" compid="COMPID" fixversion="4.2" heartbeat="60">
  <Persister>
    <JournalingPersister id="jpl" baseDirectoryName="./data/persist/${session}"/>
  </Persister>
</Session>
```

This will create persistence files whose names start with *FIX1~CTRCOMPID~COMPID*- in the *data/persist/* directory.

## 3.2. Logging

The Catalys Node has a highly configurable logging sub-system. It can be configured to use:

- High Performance Logger (HPL), which produces log files in binary format and is often used in conjunction with the Catalys MIS and Dashboard
- Logback, which produces text-based log files and console output
- A combination of both HPL and Logback

### 3.2.1. Choosing a Logging Implementation

When using only HPL-based logging, no log messages appear on the console and no human-readable log files are created. Instead the logs are written to disk in binary format. Apart from the performance benefit of logging binary data instead of text, using HPL also enables the real-time streaming of log data to the MIS for viewing in the Dashboard. Please refer to the Catalys MIS documentation for more details.

When HPL is disabled, Logback is implicitly enabled and standard text-based logging will be produced. Log messages can be output to the console and to one or more files. The format and level of the logs are configured via the *logback.xml* configuration file.

Mixed-mode operation of the two logging implementations is possible. This is most common when you require text-based logging but also wish to stream logs to the MIS. To use mixed-mode operation, HPL must be disabled in *HighPerformanceLogger.conf* and the `HighPerformanceLoggerAppender` must be configured in *logback.xml*. Note that in mixed-mode operation, even though HPL is "disabled", the HPL configuration is still applicable (e.g. the directory where HPL logs are created). A complete [configuration](#) that uses mixed-mode logging is shown below.

### 3.2.2. HPL Configuration and Operation

HPL can be enabled or disabled in the *HighPerformanceLogger.conf* configuration file, whose directory location must appear in the classpath for the Node instance using it. If you used the installer this is located in the `<CATALYS_NODE_INSTANCES>/<INSTANCE>/conf` directory. The configuration file has the following parameters:

Parameter	Description
enabled	Controls the state of the High Performance Logger.  When true, HPL is enabled. When false, HPL is disabled.  Defaults to true.
logAppenderClassName	Choose between the asynchronous or synchronous logger.  Defaults to asynchronous.
directory	The directory where the log files are written.  Defaults to the working directory of the application.
level	The level of logging.  Can be ALL, TRACE, DEBUG, INFO, WARN, ERROR, OFF.  Defaults to INFO.
maxFileLength	The maximum length of the log file before it is rolled over.  Defaults to 2GB.
maxTotalFileLength	The following property specifies the maximum total file length (in bytes) for all high performance log files before deleting the oldest files. Log files will be deleted until the total current length of all existing high performance log files plus the maxFileLength is equal or below the total length cap. A value of 0 disables the cap.



## Configuration

Parameter	Description
	Defaults to 0.
minBufferSize	<p>The minimum number of entries in the asynchronous appender's buffer. The buffer is allocated in memory at startup to avoid garbage created as the result of dynamic allocations at runtime. The actual buffer size is the next power of two greater than the specified minimum. Increasing the buffer size decreases the probability that a logger will block when trying to produce a log entry.</p> <p>Defaults to 10,000 entries</p>
minBufferEntrySize	<p>The minimum number of bytes allocated to each buffer entry in the asynchronous appender's buffer. The memory for each buffer entry is allocated at startup to avoid garbage created as the result of dynamic allocations at runtime. Individual buffer entries grow as required but never release any additional memory acquired. Increasing the buffer entry size decreases the probability of additional memory allocation at runtime.</p>
logWriterBatchSize	<p>The number of bytes allocated to the writer's batch buffer. The log writer uses a batch buffer to avoid excessive writes to disk under high throughput conditions. Optimal results usually occur when the batch size is a multiple of the underlying device's block size. Batching by the log writer only occurs when it detects that there are multiple log entries queued up waiting to be written. This value must not be larger than the configured 'maxFileLength'.</p> <p>Defaults to 8MB.</p>
beaconInterval	<p>Specifies how long to wait (in seconds) of no log messages being logged before a beacon log message is logged. A value of 0 disables beacon logging.</p> <p>Defaults to 0.</p>



### Caution

It is recommended that each Node instance use its own set of logging configuration files.

## Viewing Binary HPL Files

HPL log files are created in the current working directory by default and are named *camerontec.log.n*, where *n* is a sequential number starting from 1. The binary logs can be viewed in human-readable format using any of the following classes:

- [TailingHighPerformanceLogPlayer](#)
- [OfflineHighPerformanceLogPlayer](#)
- [ViewLogs](#). This class is designed to be used from the command line. Example scripts are provided in the HPL [template](#). See `viewlogs[.sh|.bat]`.

If using the installer, there is a *view-hpl-logs.[sh|bat]* script located in the `<CATALYS_NODE_INSTANCES>/example/scripts` directory.

## Configuration Variables

HPL allows environment variables and Java system properties to be referenced from the configuration file.

To set the directory path using an environment variable, first create the variable on the system and then reference the variable in the HPL configuration file using the syntax `${env:VariableName}`. For example:

```
directory = ${env:HPL_Directory_Path}
```

To set the directory path using a Java system property, add the following to your startup script:

```
java -DHPL_Directory_Path="../../HPL_Logs/" ...
```

Reference the Java system property in the configuration file using the syntax `${system:PropertyName}`. For example:

```
directory = ${system:HPL_Directory_Path}
```

## Archiving HPL Log Files

HPL log files should be regularly archived in order to avoid excessive disk space consumption. If [maxTotalFileLength](#) is not configured to limit the size of HPL log files, a custom script or other manual procedure can be implemented to archive and/or purge older logs.

If HPL log streaming is enabled, the log files are replicated to the MIS. Archiving and purging HPL log files on the Node host will also decrease disk space consumption on the MIS host.

## HPL Template

A template demonstrating the High Performance Logger can be found in *templates/HighPerformanceLogging*.

### 3.2.3. Logback Configuration

Logback is configured using a *logback.xml* file, whose directory must be present in the classpath of the Node. If multiple files with this name are present in the classpath, only the first one will be effective. Refer to the [Logback Configuration](#) documentation for complete details on this file.

We bundle two *logback.xml* configuration files with the Node. If using the installer, the file is located `<CATALYS_NODE_INSTANCES>/example/conf`. A simpler configuration file is located in the *resources* directory of the Node distribution. It logs to standard output at INFO level.

Logback allows Java system properties to be referenced from the configuration file. For example you may want to set a unique log file name for each Node instance. To do this, add a -D Java system property to your startup script:

```
java ... -DLOG_FILE_NAME="catalys-broker.log" ...
```

To refer to this value in *logback.xml*, use the `${LOG_FILE_NAME}` replacement parameter.

## Logback Appenders

Log messages are output by Logback using appenders. An appender definition consists of an appender class as well as the individual loggers which it creates. Each individual appender must have a unique name. For a full list of Logback logging appender classes, refer to the [Appenders chapter in the Logback manual](#). The [example](#) below demonstrates how to configure the following Logback appenders:

- ConsoleAppender
- FileAppender
- RollingFileAppender

## Logger Names

In general, most Java classes in the Node have their own logger, whose name is the complete class name. For example, the `CatalysServer` class logs messages to a logger named `com.camerontec.catalys.server.CatalysServer`.

Sessions and components (i.e. message processors) use a different naming scheme:

`CameronTec.Session.<session_name>`

`CameronTec.Component.<component_type>.<component_id>`

## Log Format

The format of Logback strings is highly configurable. See the examples below and the [Logback](#) documentation for more detail, in particular the [Conversion Words](#) section.

## Logback Configuration via JMX

Logback supports reconfiguration from the default configuration file. See the [JMX Configurator](#) for more information.

## Example Logback Configuration

This example demonstrates how to:

- Log inbound and outbound FIX messages to individual files, and all FIX messages to a combined file
- Log DEBUG messages to a separate file
- Use the `HighPerformanceLoggerAppender` to generate binary logs that can be streamed to the MIS (mixed-mode logging)

```
<configuration>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <!-- deny all events with a level below INFO, that is TRACE and DEBUG -->
    <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
      <level>INFO</level>
    </filter>
    <encoder>
      <pattern>
        %date %-5level [%thread] %logger - %marker%msg%n
      </pattern>
    </encoder>
  </appender>

  <appender name="SESSION1_FIX" class="ch.qos.logback.core.FileAppender">
    <file>session1_fix.log</file>
    <encoder>
      <pattern>
        %date %-5level [%thread] %logger - %marker%msg%n
      </pattern>
    </encoder>
  </appender>
```

## Configuration

```
</appender>

<appender name="SESSION1_FIX_IN" class="ch.qos.logback.core.FileAppender">
  <file>session1_in.log</file>
  <encoder>
    <pattern>
      %date %-5level [%thread] %logger - %marker%msg%n
    </pattern>
  </encoder>
</appender>

<appender name="SESSION1_FIX_OUT" class="ch.qos.logback.core.FileAppender">
  <file>session1_out.log</file>
  <encoder>
    <pattern>
      %date %-5level [%thread] %logger - %marker%msg%n
    </pattern>
  </encoder>
</appender>

<appender name="DEBUG" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>debug.log</file>
  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
    <!-- daily rollover -->
    <fileNamePattern>debug.%d{yyyy-MM-dd}.log</fileNamePattern>
    <!-- keep 30 days' worth of history -->
    <maxHistory>30</maxHistory>
  </rollingPolicy>
  <encoder>
    <pattern>
      %date %-5level [%thread] %logger - %marker%msg%n
    </pattern>
  </encoder>
</appender>

<appender name="HPL"
  class="com.camerontec.catalys.util.logger.logback.HighPerformanceLoggerAppender">
  <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
    <level>INFO</level>
  </filter>
</appender>

<logger name="CameronTec.Session.CTPTYCOMPID_OURCOMPID" level="info">
  <appender-ref ref="SESSION1_FIX"/>
</logger>

<logger name="CameronTec.Session.CTPTYCOMPID_OURCOMPID.in" level="info">
  <appender-ref ref="SESSION1_FIX_IN"/>
</logger>

<logger name="CameronTec.Session.CTPTYCOMPID_OURCOMPID.out" level="info">
  <appender-ref ref="SESSION1_FIX_OUT"/>
</logger>

<logger name="CameronTec" level="debug">
  <appender-ref ref="DEBUG"/>
</logger>
```

```
<logger name="com.camerontec.catalys" level="debug">
  <appender-ref ref="DEBUG"/>
</logger>

<root level="info">
  <appender-ref ref="STDOUT"/>
  <appender-ref ref="HPL"/>
</root>
</configuration>
```

## Logback Template

A template demonstrating Logback can be found in *templates/AdvancedLogging*.

### 3.2.4. FIX Message Logging

By default, the Node logs all inbound and outbound FIX messages at INFO level. There are three `Application` attributes which control FIX message logging for all sessions:

- `printMessages`

If true, all FIX messages are logged at INFO level.

- `printAdminMessagesOnly`

If true, only administrative messages (login, logout, heartbeat, sequence reset, test request and resend request) are logged, regardless of the value of `printMessages`.

- `printFormattedMessages`

If true, a brief message summary showing the sequence number and message type will be logged in addition to the entire message. For example:

```
2015-08-12 11:47:20,045 INFO CameronTec.Session.BROKER_CLIENT.in \
    < 2 NewOrderSingle (8=FIX.4.49=27035=D56=BROKER49=CLIENT...)
```

## Logger Names

Messages are logged by each session using loggers with the following naming scheme:

Inbound: `CameronTec.Session.<session_name>.in`

Outbound: `CameronTec.Session.<session_name>.out`

The `<session_name>` is composed of the `CompID` attributes configured for the session. Consider the following configuration:

```
<Session compid="A"
```

```
subid="B"  
locationid="C"  
counterpartycompid="D"  
counterpartysubid="E"  
counterpartylocationid="F" ...>
```

This session's inbound logger will be named `CameronTec.Session.A.B.C_D.E.F.in`

## Manually Logging FIX Messages

If you wish to print or manually log an `IFIXMessage` from a custom component, we recommend that you use the [IFIXMessage.toString\(int\)](#) method. It accepts an `int` argument that represents one of several constants that define an output format. Please refer to the [IFIXMessage](#) JavaDoc for a list and explanation of each format.

### 3.2.5. Logging API

Developers of custom Catalys Node components can take advantage of our logging API, specifically the [ILogger](#) interface, which derives from the standard [slf4j](#) interface.

All of the slf4j methods are implemented, except the variants which take a [Marker](#) argument. These are mapped onto the equivalent methods which do not take a `Marker` argument.

## Getting a Logger

Each class should have a logger, whose name is usually based on the class name:

```
package com.mycompany;  
...  
ILogger logger = LogManager.getLogger(SomeComponent.class);
```

This will obtain the logger named `com.mycompany.SomeComponent`.

Or the logger name can be a concatenation of the class and a unique identifier:

```
package com.mycompany;  
...  
String id = AttributeUtility.getStringAttribute("id");  
ILogger logger = LogManager.getLogger(SomeComponent.class.getName() + "." + id);
```

This will obtain the logger named `com.mycompany.SomeComponent.component1` when the value of `id` is `component1`.

**Note:** It's a common mistake to append with the class itself:

```
ILogger logger = LogManager.getLogger(SomeComponent.class + "." + id); // Wrong
```

This will obtain the logger named `SomeComponent.component1`, which no longer has the package hierarchy information.

## Using the Logger

There are logging methods corresponding to each logging level: `error`, `warn`, `info`, `debug` and `trace`.

The simplest calls are the methods which take a `String` argument:

```
logger.info("Primary Server: " + myServer + " is releasing the lock");
```

There are method calls which take an `Exception` and print the associated `stacktrace`:

```
logger.warn("Exception reading file", ioException);
```

The interface also provides methods to lookup a message string from a resource bundle:

```
logger.errorWithLookup(myResourceBundleCode, myParameter1, myParameter2);
```

Refer to the [ILogger](#) documentation for the complete set of available methods.

## 3.3. JMX Configuration API

### 3.3.1. Introduction

The Catalys Node provides a JMX API for editing the configuration of a running Node. This API allows clients to write applications that can remotely view and modify any aspect of the Node's configuration.

### 3.3.2. Management Infrastructure

The Catalys Node JMX configuration API leverages the Catalys management infrastructure. This means the Local Management Agent (LMA) and the Management Agent (MA) can be used to collate the JMX interfaces of a number of Nodes running across a number of different machines.

## JMXManagementService

In order to be manageable via JMX the `JMXManagementService` must be declared in the application's startup XML configuration. See [JMXManagementService](#) for more information.

```
<Services>
  <JMXManagementService id="JMXManagementService"/>
```



```
</Services>
```

For more information about the Catalys management infrastructure please see [Monitoring via JMX](#).

Please note that using the management infrastructure changes the names of MBeans. Both the LMA and the MA augment the names of forwarded MBeans with additional information so that multiple instances of Nodes running on either the same host or on different hosts can be distinguished. This means that client code written against the Node process, or against the LMA, or against the MA will not be exactly compatible. For this reason we advise customers to write their applications for the MA, the most flexible and scalable solution.

### 3.3.3. Configuration versus Runtime

Catalys Nodes make a distinction between changes to the configuration of the Node and changes to the runtime. All changes made using the configuration API are changes to the configuration only. These changes impact only the in-memory representation of the configuration maintained by Catalys. They will have no impact on the runtime state of a Node until a special `reload` operation is invoked.

Once `reload` has been invoked, any changes made to the configuration will be validated, and if the changes are safe they will be applied to the runtime.

### 3.3.4. Reload

On reload, the Node evaluates the safety of all changes made to the configuration. A change is only considered safe if it has no impact on a running session. More precisely, it is forbidden to make a change to a session, any of the session's contents, or anything that the session or its contents depend on, unless the session is disconnected, blocked, and its session manager is deactivated.

If the Node determines that any of the changes made to the configuration are unsafe, the whole reload operation will be aborted. Changes are either all applied or not applied at all. Once a reload has been aborted the configuration will still contain the changes made previously, allowing users to either adjust their desired changes, disconnect and block sessions if desired, or perhaps to revert the whole set of changes.

The API provides operations to allow sessions to be easily placed in a safe state, and to be easily returned to a runnable state. These operations are called `enableSessionReload` and `disableSessionReload`. Invoking `enableSessionReload` will block and disconnect a session and deactivate its session's session manager. Invoking `disableSessionReload` will activate a session's session manager and unblock the session.

### 3.3.5. Refresh

Some changes to attributes of the configuration are not able to be reloaded into runtime. Attributes for which this is the case are referred to as "non-refreshable". For example: the `customComponentScanningPath` attribute of the `Application` element.

Each configuration attribute has a `refreshable` property which indicates whether it can be changed at runtime or not.

Each configuration element has a `hasNonRefreshedChanges` attribute and `nonRefreshedChanges` operation, which indicate changes made to non-refreshable attributes. These are reported accurately unless there is a manual configuration reload (XML reload via the CLI), in which case the refreshed changes are cleared and can no longer be reported through the JMX API.

### 3.3.6. Undo

The configuration API provides an explicit operation for undoing changes. This will revert any changes to the configuration that have not already been applied to the runtime.

### 3.3.7. Persistence

Changes made using the configuration API are persisted to XML after each successful reload. Before updating the XML configuration file, the Node persists a timestamped backup of the existing configuration file (from before the changes are applied). This allows users to easily fall back to old configurations and also provides a record of the changes made over time.

In addition, there is an explicit `persist` operation that allows users to trigger persistence. This can be used for making changes to the configuration that should only be applied on the next restart.

### 3.3.8. MBeans

The configuration API is presented to clients in the form of MBeans. Anything that is configured will expose an MBean. An MBean will be registered for the Application, for each Session, for each configured message processor, etc.

## Naming Scheme

The configuration API names its MBeans according to [Sun's JMX Best Practices Guidelines](#). It is highly recommended that clients interested in the naming scheme read these best practices, in particular [Object Name Conventions](#) and [Object containment](#).

The configuration API MBeans can be distinguished from other Node MBeans by looking for the `Configuration` qualifier in the object name type. For example, the type of the Session MBean is `Configuration.Application.Session`. All configuration MBeans contain this qualifier.

Note that object names will change depending on whether you are accessing the MBeans by connecting directly to the Node, or whether you are connecting to the LMA or MA. The LMA and MA add additional qualifiers to allow multiple Node instances to be collated. For example, the MA adds "Host" to the type. For this reason we advise customers to write their applications for the MA, the most flexible and scalable solution.

## Attributes

Every MBean exposes a set of attributes identical to the attributes of its corresponding XML element. For example, the `SampleMessageSource` MBean exposes an `interval` attribute of type integer, representing the time between sending messages.

In addition to their configuration attributes, some MBeans have additional attributes used to list their relationships with other MBeans. For example, the `SessionManager` MBean has a `sourceMessageProcessors` attribute that contains an array of JMX Object Names referring to the session manager's source message processors.

## Operations

In addition to its attributes, every MBean exposes a set of operations. The operations are used for adding and deleting configuration elements, for reloading changes, for undoing changes, and for various other purposes.

## Types of MBeans

There are many different MBean types, including:

MBean Type	Relationship Attributes	Notable Operations
Configuration.Application		reload, undo, saveConfiguration, addSession, addService
Configuration.Application.-Session	childToFIXMessageFactory, childFromFIXMessageFactory, childToFIXCompression, childFromFIXCompression	reload, undo, markForDelete, addSessionPersister, copySession
Configuration.Application.-Session.SessionManager	sourceMessageProcessors, listenerMessageProcessors	markForDelete, addMessageProcessor, addSelector
Configuration.Application.-MessageProcessor	messageProcessors	markForDelete, addMessageProcessor, addSelector

### 3.3.9. Interactively Exploring the API

The configuration API can be interactively explored using a JMX client such as JConsole. It is highly recommended that clients interested in using the API use JConsole or another generic client application to become familiar with it.

A word of warning about JConsole. It does not allow users to set attributes to `null` or `unset`, and it does not allow users to pass null as an argument to operations. In addition, it does not allow the modification of complex types, although they can be viewed.

### 3.3.10. Example

There is an example client application provided in the *templates/JMXConfigurationClient* directory. This sample application simply adds a new session (with a persister, connection and message processor) to an application.

The source code for the application is located here: *templates/src/com/camerontec/catalys/server/test/SampleJMXConfigurationClientWhichAddsNewSession.java*

## 3.4. Encryption of Attributes

Certain attributes in the XML configuration file can be encrypted, typically used for passwords or host names for example.



### Important

The following attributes in the configuration file cannot be encrypted:

- Any attributes of XML elements which do not contain an id (exception: `<Session>`)
- RulesPack and all nested elements
- Any attributes which are part of a connection point: compid, subid, locationid (and counterparty's)
- Attributes of type Boolean



### Note

Your JVM's JCE policy files might have to be updated to include Unlimited Strength Jurisdiction.

These can be downloaded here:

[Properties](#)

- [JCE for java 6](#)
- [JCE for java 7](#)

## Configuration

- [JCE for java 8](#)
- From Java 9 this can be done via configuration:

```
/conf/security/java.security

crypto.policy=unlimited
```

### 3.4.1. Encrypt an Attribute

To encrypt an attribute a simple pattern must be followed `encryptme://{value}` and the value will be encrypted on the next restart of the node or `config_reload`.

For example:

```
[...]
<Session counterpartycompid="MARKET" compid="BROKER" fixversion="4.4" heartbeat="60">
  <Connections>
    <SocketConnection hostname="encryptme://{localhost}" port="2002" id="sc1" />
  </Connections>
  <SessionManager clientConnectionManager="com.camerontec.catalys.server.test.UserPassClientConnectionManager">
    <Properties id="clientConnectionmanagerProperties">
      <Property name="password" value="encryptme://{pass123}" />
      <Property name="username" value="jon" />
    </Properties>
  </SessionManager>
</Session>
[...]
```

### 3.4.2. Decrypt an Attribute

Once an attributes has been encrypted it will be written back to the configuration file with the pattern `encrypted://{encryptedvalue}`.

For example:

```
[...]
<Session counterpartycompid="MARKET" compid="BROKER" fixversion="4.4" heartbeat="60">
  <Connections>
    <SocketConnection hostname="encrypted://{dDFE8XvfRRJSej6MoCMKjYScEZDYjpYN}" port="2002" id="sc1" />
  </Connections>
  <SessionManager clientConnectionManager="com.camerontec.catalys.server.test.UserPassClientConnectionManager">
    <Properties id="idN65561">
      <Property name="password" value="encrypted://{tkksJ0ZxZs5UgafqAVGzwQ==}" />
      <Property name="username" value="jon" />
    </Properties>
  </SessionManager>
</Session>
[...]
```

# Chapter 4. Operations and Monitoring

## Table of Contents

4.1. Node Lifecycle .....	59
4.1.1. Introduction .....	59
4.1.2. Starting the Node .....	60
4.1.3. Starting FIX Sessions .....	60
4.1.4. Stopping FIX Sessions .....	60
4.1.5. Resetting FIX Sessions .....	60
4.1.6. End-of-Day Processing .....	61
4.1.7. Stopping the Node .....	61
4.1.8. Archiving Logs .....	61
4.2. Monitoring the Node .....	61
4.2.1. Introduction .....	61
4.2.2. Command-Line Interface .....	61
4.2.3. Remote Command-Line Interface .....	70
4.2.4. Custom CLI Commands .....	73
4.2.5. Monitoring via JMX .....	73
4.2.6. Monitoring via the Dashboard .....	78
4.2.7. Developing a Custom Monitor Class .....	78
4.3. Configuration Model and Runtime Changes .....	78
4.3.1. Introduction .....	78
4.3.2. Configuration Model vs. Runtime Model .....	78
4.3.3. Dynamic Configuration .....	79

## 4.1. Node Lifecycle

### 4.1.1. Introduction

All production FIX environments need to start, stop, reset and shutdown on a strict, well-known schedule. In addition to the scheduling of the FIX engine and its sessions, logs must be periodically archived and persistent data should be recycled.

Most FIX environments operate on a daily or weekly lifecycle, but any time period is permitted. The Node supports most of these lifecycle tasks natively using the built-in [Scheduler](#), but there are other ways to perform some of the tasks.

## 4.1.2. Starting the Node

One lifecycle task that the Scheduler cannot perform is starting the Node process. This must be performed by a process outside of the Node. In Unix-based environments this is normally performed using cron. In Windows environments this is normally performed using a Scheduled Task. In either case, the job or task should be configured to execute the Node's [startup script](#).

## 4.1.3. Starting FIX Sessions

After the Node has started, sessions will be scheduled to connect and logon with their counterparty. A listening session will allow the counterparty to connect and logon at this time; an initiator will connect and attempt to logon to the counterparty. This can be scheduled using a `StartTask` in the Scheduler of the Node. Sessions can also be manually started via the Node's command-line interface (CLI) by issuing the [s\\_connect](#) command.

## 4.1.4. Stopping FIX Sessions

Once a FIX session is established it will be scheduled to logout and disconnect from the counterparty. It is highly recommended that FIX sessions perform a graceful logout sequence rather than abruptly disconnecting the socket. This gives both sides time to send any final messages before logging out. A graceful logout can be scheduled using a `StopTask` in the Scheduler of the Node. Sessions can also be manually stopped by issuing the [s\\_disconnect](#) CLI command.

## 4.1.5. Resetting FIX Sessions

A FIX session reset is normally performed after a `StopTask` has executed and before the next `StartTask` executes. This ensures that both sides of a FIX session logon with the starting sequence number #1, and it protects a counterparty from accidentally resending a batch of messages from the previous day, week or session. A session reset can be scheduled using a `ResetTask` in the Scheduler of the Node. Sessions can also be manually reset by issuing the [s\\_reset](#) CLI command.



### Caution

Resetting a FIX session that is connected will abruptly terminate the session and reset the inbound and outbound sequence numbers to #1. This is not recommended and should only be performed if agreed on with your counterparty.

## 4.1.6. End-of-Day Processing

"End-of-Day" processing invokes end-of-day procedures on all configured message processors. This usually involves purging or compacting persistent data. There are certain features in the Node that store persistent data but custom message processors could store persistent data as well. EOD processing can be scheduled using the `EndOfDay` task in the Scheduler of the Node. This can also be manually triggered by issuing the [s\\_end\\_of\\_day](#) CLI command.

## 4.1.7. Stopping the Node

Periodically stopping the Node is recommended but not required. Restarts are only required when making certain configuration changes. Restarting will clear out any data stored in memory and entirely reset the FIX environment. Stopping a Node can be scheduled using the `ShutdownTask` the Scheduler of the Node. It can also be manually performed by issuing the [exit](#) CLI command.

## 4.1.8. Archiving Logs

Another common lifecycle task is archiving and/or purging old log data. This is normally configured to adhere to the file archiving policies at your firm. The Node can produce two types of log data: text and binary.

Text-based logs are produced using [Logback](#) and are configured via *logback.xml*. The archival of these logs can be manually managed, or you can use Logback's built-in [RollingFileAppender](#).

Binary logs are produced using [High Performance Logger](#). HPL logs are configured via the *HighPerformanceLogger.conf* file. See the HPL file archiving section [here](#).

# 4.2. Monitoring the Node

## 4.2.1. Introduction

There are numerous ways to monitor a running Node: MIS/Dashboard, command line, JMX API, custom monitor class, log files, etc. There are several different aspects of the Node that can be monitored: server and FIX session events, application events, memory usage, disk space, latency and more.

## 4.2.2. Command-Line Interface

The Node's command-line interface (CLI) allows users to issue commands directly to a running Node through a console prompt. The CLI is available from the console where the server was started, or remotely via JMX. The system is designed to allow other custom components to extend the command line.



## Configuring the Console

By default the CLI is available through the console window where the Node is running. This option can be disabled in the XML configuration file:

```
<Application console="none">
```

## Command Results

When a command is issued successfully, the console will output three parts: header, separator and results:

```
Header1           Header2
-----
Results_for_Header1 Results_for_Header2
```

When a command does not execute successfully, the command name and a description of the problem will be printed:

```
Command Description
-----
s_list Failed to find a matching session. Session: #444
```

## Command Completion

Due to the length of some command names, the CLI provides automatic command completion for most commands (except for `exit`). This saves typing large commands, for example:

```
> s_out_seq_number #10 1001
```

Can be entered as:

```
> s_out #10 1001
```

The general rule is that only the first three to four characters need to be typed in order to uniquely identify the correct command. If for any reason the console doesn't understand the command, it will print a description of the error in the command results.

## Listing and Getting Help on Commands

To get a complete list of available commands enter `? or help`.

To get help for a specific command enter `? <command>`. For example:

```
> ? s_list
Command Description
-----
s_list <all | session>: list a session
```

The arguments listed in the description follow these rules:

- <arg> means that the argument should be substituted with literal text
- [arg] means that the argument is optional
- | indicates that the arguments are alternatives

## Reserved Characters

Certain characters are reserved for use by the command-line processor:

Character	Usage
*	Wildcard
all	Wildcard
:	Demarcates the boundary between an application ID and a session name when specifying a session
.	Demarcates the boundaries between IDs when specifying a session
/	Strictly matches the session with empty parameter (as opposed to matching sessions regardless of the value of this parameter)

### 4.2.2.1. Default Commands

Some commands are only available if a certain feature is configured. The following commands, however, are always available via the CLI regardless of which features are configured.

Command	Arguments	Description
? or help	<command   all>	Get help on a command or list all commands
exit		Shut down the Node
exit_gracefully		Logs out all sessions and shut down the Node
config_reload		Dynamically reload configuration into the runtime of the Node.

Command	Arguments	Description
		If there are modified sessions then these need to be "reloadable" for this command to succeed. A session is "reloadable" if it is stopped (disconnected and de-activated) and blocked from receiving any further connections. This can be achieved using the <a href="#">s make reloadable</a> command. Sessions can be restored to their previous state using <a href="#">s unblock and activate</a> .
level_of_logging	<logger name   all> <level>	Set the level for a given logger
license_reload		Reload the current license file
logger	<logger name   all> <level>	List the level of a given logger
memory		Show used and available memory
ping		Confirms that server is alive
threads	<stackTrace>	List all the threads in the system
version		List version and build information

#### 4.2.2.2. Component Commands

Many Node components implement a simple [IComponent](#) interface that can be accessed from the command line. This allows the user to close and activate/de-activate individual components.

Command	Arguments	Description
c_activate	<name   all>	Activate a component
c_de_activate	<name   all>	Deactivate a component
c_list		List all of the registered components in the system
c_properties	<name   all>	List the components properties
c_refresh	<name   all>	Refresh a component

### 4.2.2.3. Session Commands

Session commands can be issued for a specific session or for all sessions. Most session commands require that you type the command followed by `all` or a `session_id` argument. The `session_id` argument can be specified one of two ways: hash identifier or the full target/sender ID combination.

The following is an example of a range of sessions with different Target/Sender Comp, Sub, and Location IDs:

```
> s_list all
#   TargetCompId TargetSubId TargetLocation  SenderCompId SenderSubId SenderLocation
---
73  SellSide1
161 SellSide4      subid4      locationid4  BuySide4
44  SellSide9
5   SellSide7      SellSubid7 SellLocationid7 BuySide7      BuySubid7      BuyLocationid7
```

#### Hash Identifier

From the above example, you can see the far left column has a unique hash ID that can be used at the command line to uniquely identify a session. Here are some examples:

```
> s_list #161
#   TargetCompId TargetSubId TargetLocation  SenderCompId SenderSubId SenderLocation
---
161 SellSide4      subid4      locationid4  BuySide4

> s_in #44 100
#   TargetCompId SenderCompId Status InSeqNo OutSeqNo
--
44  SellSide9      BuySide9      DOWN      100      0
```

#### Full Name identification

Sessions can be uniquely identified by a set of six parameters. The `SessionCommandLineProcessor` uses the following syntax to uniquely define a session:

`TargetCompId.TargetSubId.TargetLocationId/SenderCompId.SenderSubId.SenderLocationId`

To see the full names of all configured sessions, issue the `s_fullname` command. Here is the same list of sessions from above shown via this command:

```
> s_fullname
#   Full Name
---
73  SellSide1/BuySide1
161 SellSide4.subid4.locationid4/BuySide4
44  SellSide9/BuySide9..BuyLocationid9
5   SellSide7.SellSubid7.SellLocationid7/BuySide7.BuySubid7.BuyLocationid7
```

Here are some examples of identifying a session using its name:

```
> s_s SellSide7/BuySide7
# TargetCompId SenderCompId Status InSeqNo OutSeqNo
-----
5 SellSide7      BuySide7      DOWN    0          0

> s_s ../locationid4
# TargetCompId SenderCompId Status InSeqNo OutSeqNo
-----
157 SellSide4     BuySide4     DOWN    0          0

> s_s ../BuyLocationid7
# TargetCompId SenderCompId Status InSeqNo OutSeqNo
-----
105 SellSide7     BuySide7     DOWN    0          0
```

As you can see from the example, the full name of the session is not required to uniquely identify it. In fact, if two sessions have the same TargetLocationId and the only information entered at the command line is the TargetLocationId, both sessions will be selected.

## Session Command Reference

Command	Arguments	Description
s_block	<all   session>	Blocks connections on this session
s_connect	<all>   <session> <connection #>	<p>Enable any automatic connection mechanism that has been previously disabled via <code>s_disconnect</code>. Note: All inactive sessions will be ignored.</p> <p>An optional connection number starting from 1, indicating the index amongst all available connections, may be specified. If that's the case, an attempt to establish a connection will be performed once, and in case of failure, the default behaviour will be used (i.e. using the first connection only, according to configuration order).</p>
s_disconnect	<all   session> [<reason >]	<p>Disconnect a session by sending a logout message. This also disables any automatic reconnect mechanism.</p> <p>If optional reason is specified it is included in Text (58) tag in Logout</p>

Command	Arguments	Description
		message. Note: All inactive sessions will be ignored.
s_drop	<all   session>	Force a session to drop immediately. Note: All inactive sessions will be ignored.
s_end_of_day	<all   session> [<parameter>]	<p>Sequentially invokes the end of day procedure on all message processors in the message processing pipeline of the selected session(s). This usually involves purging or compacting data in persistent files.</p> <p>The optional parameter is used in a component specific manner by the message processor executing the procedure. For those components that use MarketMirror storage, this parameter is interpreted as a clearing condition specified using the Rules Expression Language. It is only needed if a value different from that specified by the component's <code>endOfDayClearingCondition</code> attribute is required.</p> <p>By implementing the <a href="#">IFIXMessageProcessor</a> interface and its <a href="#">performEndOfDayProcedure</a> method, custom message processors can hook into this mechanism and provide their own end-of-day logic, if necessary.</p>
s_force_reload	<all   session>	Forces one or more sessions into the reloadable state, loads changes to the XML configuration into the Node's runtime, then restores the previous state of the session(s).

Command	Arguments	Description
s_full_name		List the full name of each session
s_in_seq_number	<all   session> <seq_number>	<p>Set the inbound sequence number of a session. This is the next expected sequence number from a counterparty. Note: All inactive sessions will be ignored.</p> <p>To update sequence numbers of an active session:</p> <ul style="list-style-type: none"> <li>• Block the session(s_block)</li> <li>• Disconnect the session(s_disconnect)</li> <li>• Update sequence numbers</li> <li>• Unblock session(s_unblock)</li> <li>• Connect the session(s_connect)</li> </ul>
s_list	<all   session>	List all the sessions in the system
s_lock	<all   session>	Locks connections on this session. A locked session acts like a blocked one. The difference is that the locked state is not persisted and only valid during the life of the FIX server.
s_make_reloadable		Puts one or more sessions into the reloadable state. That is, disconnects the session(s), deactivates their session manager(s) and blocks the session(s) from receiving any further connections.
s_out_seq_number	<all   session> <seq_number>	<p>Set the outbound sequence number of a session. The will be the sequence number of the next message sent to a counterparty. Note: All inactive sessions will be ignored.</p> <p>To update sequence numbers of an active session:</p>

Command	Arguments	Description
		<ul style="list-style-type: none"> <li>• Block the session(s_block)</li> <li>• Disconnect the session(s_disconnect)</li> <li>• Update sequence numbers</li> <li>• Unblock session(s_unblock)</li> <li>• Connect the session(s_connect)</li> </ul>
s_reload	<all   session>	Loads changes to the XML configuration of one or all sessions into the Node's runtime. The specified sessions must be in a reloadable state for this command to succeed.
s_reprocess	<session> <beginSeq> [<endSeq>]	<p>Reprocess incoming messages. Available on sessions having a persister configured to persist input messages (e.g. <code>JournalingPersister</code> configured with <code>persistInputMessages</code> set to <code>true</code>).</p> <p>For example, if orders with sequence numbers from 100 to 105 were received but lost by the internal OMS, you could issue <code>s_reprocess &lt;session&gt; 100 105</code>. This would scan the persister for messages between 100 and 105, inclusive, and send messages (except heartbeat, test request, resend request, and sequence reset) to <code>ListenerMessageProcessors</code>.</p> <p>If <code>endSeq</code> is not specified, only one message will be reprocessed: <code>beginSeq</code>. Note: All inactive sessions will be ignored.</p>
s_resend	<session> <beginSeq> [<endSeq>]	Trigger a resend of messages from the sessions persistence as if the counterparty had requested it (with



Command	Arguments	Description
		a FIX Resend Request). Only one message is resent when only <code>beginSeq</code> is specified. Otherwise, a range of messages is sent, from <code>beginSeq</code> to <code>endSeq</code> , inclusive. Note: All inactive sessions will be ignored.
<code>s_reset</code>	<code>&lt;all   session&gt;</code>	Set the sequence numbers of a session back to 1. Note: All inactive sessions will be ignored.
<code>s_stats</code>	<code>&lt;all   session&gt;</code>	Get the current statistics of a session. This includes connection status and sequence numbers.
<code>s_unblock</code>	<code>&lt;all   session&gt;</code>	Unblocks connections on this session
<code>s_unblock_and_activate</code>	<code>[&lt;session&gt;]</code> <code>&lt;session&gt;</code>	Unblocks and activates one or more sessions. This command is used in conjunction with <code>s_make_reloadable</code> .
<code>s_unlock</code>	<code>&lt;all   session&gt;</code>	Unlocks connections on this session

### 4.2.3. Remote Command-Line Interface

The Node's Remote Command Line can execute commands on any Node instance remotely without being logged onto the server. JMX is the communication layer used between the Node and the Remote Command Line utility. It can also be so configured to authenticate the Remote Command Line utility to be authenticated before connecting.

If you used the Node installer and selected the 'Node Instances' option, the example instance is configured to startup the remote CLI. Once you start this instance use the `start-cli.[sh.bat]` script located in the `<CATALYS_NODE_INSTANCES>/example/scripts` directory.

#### 4.2.3.1. Configuring the JMX Service

To be able to communicate with an instance of a Node remotely we need to be able to connect to a JMX Service exposing the command line of that server. There are three ways to do that:

1. An instance of a Node directly
2. A LMA, bundling several Node instances on the same machine
3. A MA (MIS), bundling several LMAs across the whole network

## Automatic Detection of a Service

The Remote Command Line utility will try to connect to the specified host (parameter `-jmxHost`) on the specified port (parameter `-jmxPort`). If no host is provided, `localhost` will be used. If no port is provided, the default MA port (10000) will be used and if that connection fails, the default LMA port (10002) will be tried.

There is no default to connect to an instance of a Node as the port will be unknown to the Remote Command Line. You can configure the `JMXManagementService` in the Node to expose the command line via JMX on a particular port with the following:

```
<Services>
  <JMXManagementService
    id="JMXManagementService"
    jmxServiceURL="service:jmx:jmxmp://localhost:1500"
    alwaysKeepJMXConnectorServerRunning="true"/>
  </Services>
```

This will expose the `JMXManagementService` on port 1500. This port can then be addressed via arguments in the `RemoteCommandLine` utility. The attribute `alwaysKeepJMXConnectorServerRunning` must be `true` in order to keep the `JMXConnector` alive, even if not connected to an LMA.

## Connecting and Executing a Command

The syntax of the `RemoteCommandLine` class is:

```
java -cp "../lib/catalys-node.jar:../resources" \
  com.camerontec.catalys.server.management.client.commandline.RemoteCommandLine \
  [-jmxHost <host>] [-jmxPort <port>] [-d(ebug)] [-h <host>] [-i <instance>] \
  <command> <cmd args>
```

Parameter	Description
<code>-jmxHost</code>	The host where the JMX service is running on. This parameter is optional and <code>localhost</code> will be used if not specified.
<code>-jmxPort</code>	The port where the JMX service is listening on. The parameter is optional and the MA/MIS and LMA defaults will be used if not specified.
<code>-console</code>	Run in console mode
<code>-h</code>	The host name of the Node where the command should be execute on. This parameter is only required when connected to an MA/MIS. When connecting to a Node directly, the attribute will be ignored.

Parameter	Description
-i	The instance name of the Node the command should be executed on. This parameter is only required when connected to an MA/MIS and LMA. When connecting to a Node directly, the attribute will be ignored.
<command>	The command to be executed
<cmd args>	Possible arguments to be passed into the command executed
-?	Display help results
-d	Enable DEBUG output
-u	Username with which to connect to the Node server. This parameter is optional. If not used, Remote command line utility will not be authenticated before connecting to the Node server.
-p	Password with which to connect to the Node server. This parameter is optional and used along with -u option.

## Running in Console Mode

When the `-console` parameter is included the Remote Command Line utility will run in console mode. When successfully connected, a prompt will appear and commands can be executed against that particular instance of the Node.



### Note

To exit the Remote Command Line console, type `exit`.

## Using authenticated JMX access to connect to the server

Remote Command Line utility can be configured to connect to the Node server on which JMX authentication is required for connecting. For this, both `-u` and `-p` options need to be set correctly. If used, the Node instance to which this utility client is connecting will have to be already configured for authenticated jmx access. Catalys Node server jmx authentication configuration [here](#).

## Remote Command Line Template

A template located in `templates/src/RemoteCommandLine/` demonstrates how to use the utility.

## 4.2.4. Custom CLI Commands

It is possible to extend the list of commands by implementing the [ICommandLineProcessor](#) interface. This can be helpful for printing information related to a custom message processor or service.

See the custom [CLI](#) example in the Programming Guide.

## 4.2.5. Monitoring via JMX

### 4.2.5.1. Introduction

The Catalys management infrastructure provides a Java-based solution for scalable monitoring of the Node by utilizing [the standard management extension for Java](#), which is known as JMX. The infrastructure consists of:

- The Local Management Agent (LMA), which runs as a service on any host where a Node is running.
- The Management Agent (MA), which is a JMX management agent that connects to one or more LMAs and provides an aggregated management API of each registered Node.



#### Note

The MA is implemented in the Catalys MIS, which must be running to fully take advantage of all monitoring features.

The management API is exposed through JMX and can be browsed with any JMX-compliant monitoring console (IBM Tivoli, HP OpenView, et al.).

This section provides the specifications of the JMX API from the perspective of a third-party monitoring system which would connect to the MA. We assume that the reader is familiar with JMX, the Catalys Node and the FIX protocol.

It would be possible for any JMX-enabled application to connect directly to the LMA and even to the Node via JMX, but the third-party system would not benefit from the main features of the MA, namely:

- Convention over configuration approach (no configuration required to manage a new FIX engine)
- Discovery of new manageable applications
- Consistent naming scheme
- Persistence layer
- Ability to start processes remotely

- Scalability:
  - One central point of contact for any number of managed applications
  - Metadata-driven caching system aimed at reducing the load on managed applications
  - Distributed asynchronous model for the processing of logs data
  - Firewall-friendly

Therefore we strongly recommend that third-party systems interface with the MA.

#### 4.2.5.2. Available Features

The JMX API is meant to provide sufficient information to allow third-party and internal systems to deliver monitoring functionality for clients who elect to not use the Dashboard. Not all aspects of Catalys monitoring and management can be achieved via the JMX API though; some of them are only accessible from the Dashboard. The following table provides a comparison:

Category	Functionality	Catalys Dashboard	JMX API
Overall	Audit trail	Available	N/A
Overall	Role-based access control	Available	N/A
Server management	View server information and configuration	Available	Available
Server management	Start/stop servers	Available	Available
Server management	Command-line access	Available	Available
Server management	Startup script details	Available	Available
Server management	View HA cluster details and trigger failover	Available	Available
Session management	Sessions action (Start, stop, block...)	Available	Available
Session management	FIX Protocol admin actions	Available	Available
Log management	View log details	Partly Available	Available
Log management	Enable/disable log streaming	Available	Available

Category	Functionality	Catalys Dashboard	JMX API
Log management	View contents of logs	Available	N/A
MA management	Mount points, managed hosts	Available	Available
Events management	Subscribe to change notifications	N/A	Partly Available
Platform management	JVM information	N/A	Available
Custom Message processors	Custom attributes	Available	Available

### 4.2.5.3. JMX authentication

JMX connections can be configured to be authenticated before they exchange data. To enable the authentication, changes would have to be done for both the client side and the server side. Catalys Node acts as a client when connecting to LMA and as a server when connecting to jconsole or any other jmx client. Credentials properties file would have to be specified on the client side. Whereas, on the server side, access and password files will have to be specified.

For catalys node instance to act as a jmx server, two environment variables will have to be created. Firstly, `JMX_REMOTE_X_ACCESS_FILE`, which specifies the absolute location of the jmx access file and secondly, `JMX_REMOTE_X_PASSWORD_FILE` which specifies the absolute location of the jmx password file. For instance, environment variables can be the following: For access file:

```
JMX_REMOTE_X_ACCESS_FILE=C:\Itiviti\Catalys\Node\resources\jmxremote.access
```

For password file:

```
JMX_REMOTE_X_PASSWORD_FILE=C:\Itiviti\Catalys\Node\resources\jmxremote.password
```

The password file should have restrictive permissions. Only the owning user should have access to this file. JMX access file contains the usernames and their corresponding access levels (readwrite is required for proper operations) and JMX password file should contain the usernames and their corresponding passwords. For example,

JMX access file can contain the following:

**username readwrite**

(where username is the user for which the access level is to be set. readwrite defines access level to be granted to the user)

And JMX password file can contain the following entry:

### **username username\_password**

(where username is the user against which the JMX authentication will be done and username\_password is the password of this user)

For Catalys Node instance to act as a jmx client, *JMX\_CLIENT\_CREDENTIALS\_FILE* environment variable will have to be set. The value of this variable is the absolute path of the credentials properties file. For instance, the environment variable can be the following:

```
JMX_CLIENT_CREDENTIALS_FILE=C:\Catalys\windows\jmxHostsCredentials.properties
```

Credentials property file should be defined in below format :

### **localhost=username:password**

(where localhost is a hostname of the device where we are accessing the jmx server. And username:password is the username and password of the user which will be used to connect to the jmx server. Alternative delimiters permitted to be used in the java properties files can also be used)

## **4.2.5.4. Modes of Interaction**

### **Polling and Notifications**

From the perspective of the third-party application, the JMX API provides two approaches to receive data:

- Active polling
- Passive notification

Active polling requires that client applications should query the MA on a regular basis to receive an updated snapshot of the data.

Passive notification is enabled for some data based on technical availability and relevance. Typically, notifications are issued when the data does not change too frequently (to limit the number of notifications) and when a change in the data is likely to trigger an immediate reaction from the third-party client, such as issuing an alert.

### **Caching and Persistence**

The MA provides a sophisticated model for caching and persisting MBean attributes. For every private MBean in the managed application the MA publishes a public MBean. This public MBean is not just a

simple proxy for the private MBean, it also ensures the caching and persistence of the private MBean's attributes.

By default, each attribute has a caching of 1 second and is not persisted. Therefore if a third-party system polls the value of the attribute in the public MBean 50 times in one second, the public MBean will only forward one call to the private MBean, and will return the cached value 49 times. The fact that the attribute is not persisted means that if MA is restarted and the managed instance of the Node is not running when the MA restarts, the public MBean will still be available, but the attribute's value will be null until the connection is reestablished.

These default caching and persistence settings are overridden for selected Node MBeans based on operational relevance. Here are the possible scenarios:

- First some attributes are configured for infinite caching. This is used in cases where a change in the attribute's value is only possible if the containing MBean's identity changes. For example, a session-specific MBean will not exist if the definition of the session, such as the SenderCompID, is altered.
- Secondly, some attributes are cached every minute. Those represent values which are unlikely to change, or whose intrinsic granularity is in itself no finer than a minute.
- Then there exists a group of attributes with 5 seconds of caching. These are typically used for notifications, as described above. Polling of these attributes is discouraged, partly because it is unnecessary, but also because these attributes may be presented as complex data structure whose size may affect network traffic. Reliance on notification is the recommended approach.
- Finally, a group of attributes are cached for 1 second. These attributes are good candidates for regular polling and their size is also adequate to limit any impact on network traffic.

As a further optimization, the MA always fetches attributes by batch, and the batching of attributes is based on caching duration. Therefore, when a third-party application requests the value of attribute of one element of the group, all attributes in the group will be refreshed and cached, and a further request for another attribute in the same batch during the caching period will not generate any incremental call to the managed Node.



### Note

Caching and persistence values can be customized even further, although this can have a significant impact on performance. This customization is possible through the use of metadata descriptors, which are XML files placed in the MA's classpath at runtime and which have names that correspond to the private MBeans' names. Please contact your local support office if you believe that specific caching and persistence values should be overridden for your deployment.



See [JMX MBean Reference](#) for information about the individual MBeans that are available.

See the [Programming Guide](#) for details on how to programmatically interact with the MA.

## 4.2.6. Monitoring via the Dashboard

Please see the documentation for the Catalys MIS and Dashboard.

## 4.2.7. Developing a Custom Monitor Class

It is possible to monitor the Node in-process by developing a custom monitor class. Custom monitor classes are application level objects, so they receive events for all FIX sessions. See the [Programming Guide](#) for more details.

# 4.3. Configuration Model and Runtime Changes

## 4.3.1. Introduction

The configuration of the Node is supplied to the server when it is started, via the `-xmlconfig` command-line argument. The parameter of this argument is an XML configuration file, the elements of which are defined by *CatalysConfig.dtd*.

The XML configuration can be fully specified in this file before the Node is started; or it can be a partial or minimal configuration, which can then be modified at runtime via one of the following methods:

- Via the Dashboard
- By interfacing directly using the Catalys Node JMX interface
- By editing the original XML file and reloading it

## 4.3.2. Configuration Model vs. Runtime Model

The Node distinguishes between **configured** properties and **runtime** properties. An internal model contains a representation of the configuration of the engine which can be modified at anytime. The runtime model can be updated from the configuration model at runtime whenever it is safe to do so.

In particular, this means that sessions can be added, modified, or deleted, as long as they are **reloadable**.

## Safe Runtime Operations: Reloadable Sessions

It is only safe to reload a session if it is disconnected and blocked or locked, and its session manager is de-activated. This means that no partially processed messages can be caught in the session. When a session is in this state it is said to be **reloadable**.

The runtime model can be updated (reloaded) from the configuration model on a whole-application basis, or on a per-session basis. An application-wide reload will only succeed if all modified sessions in the application are reloadable. Unmodified sessions are ignored during an application-wide reload, therefore these do not have to be reloadable. This means that new sessions can be provisioned and activated without disrupting pre-existing active sessions.

A newly added session is reloadable. When it is loaded into runtime, it is put into the blocked state.

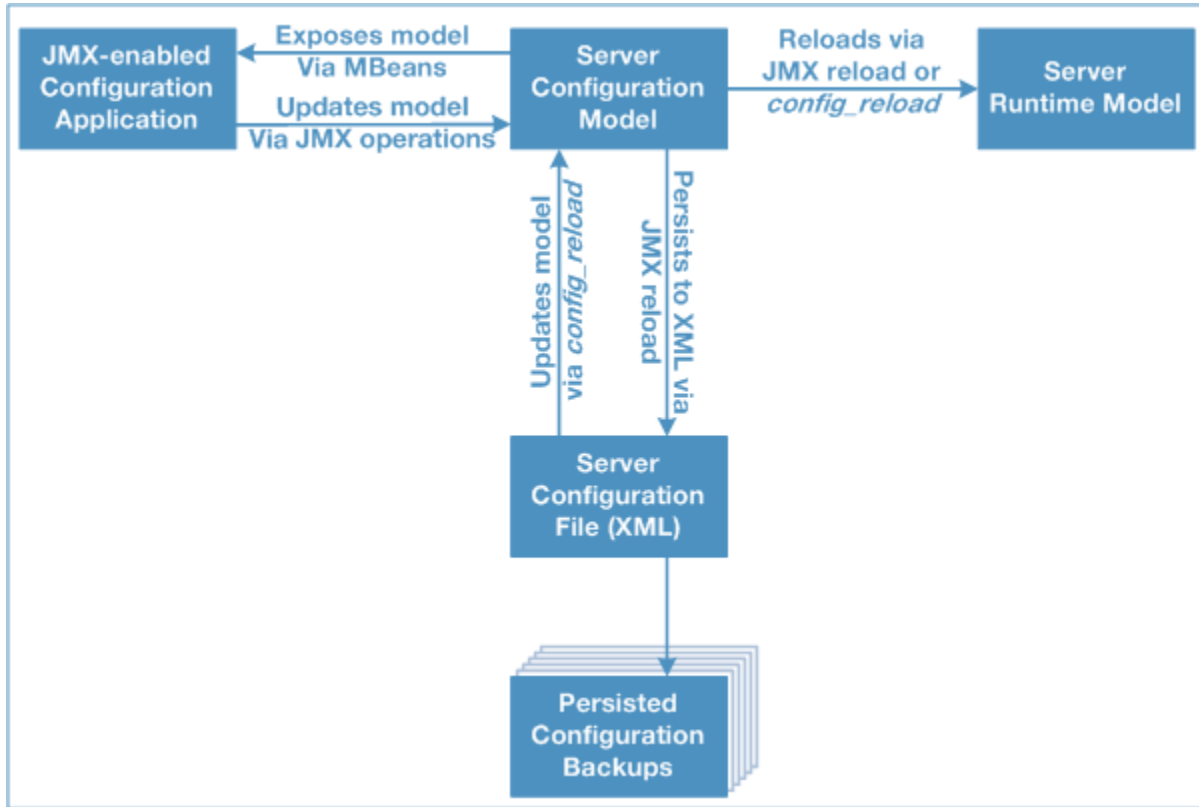
Shared configuration components (message processing templates, schedules, selectors) which are referenced by multiple sessions can only be modified and reloaded if all of the sessions that include them are reloadable. Therefore there is a trade-off between the efficiency of separating common configuration, and the flexibility of being able to reload self-contained sessions.

Sessions which share runtime components (message processors or selectors) must all be reloadable for any one of them to be reloaded. A shared runtime component is one which referenced by another component. e.g. [MessageProcessorReference](#) and [SelectorReference](#).

For configurations which use the [Link](#) element (used to route messages from one session to another), individual sessions referenced by the [SessionReference](#) element are permitted to be reloaded without all of the sessions being reloadable, but there is a chance that a message from the active session will not be able to be routed to the reloadable session. The safest solution for this scenario is to manually make all the sessions of the `Link` reloadable.

### 4.3.3. Dynamic Configuration

There are two methods available to access the server's configuration: using JMX (which the Dashboard employs) or via the XML configuration file. An overview of the two methods is shown in the following figure.



Note that the two methods cannot be used simultaneously. In particular the [config\\_reload](#) command should not be issued at the CLI while there are any pending changes resulting from JMX operations.

#### 4.3.3.1. Accessing the Model via JMX

The configuration model can be accessed and updated via JMX. Every element of the Node's configuration, from the application to sessions and components is instrumented for JMX. A number of advanced operations are also implemented, such as cloning sessions and reloading a session.

For an application to be manageable via JMX it must define the [JMXManagementService](#) in its startup XML config. To define the `JMXManagementService`, place the following lines in the XML config file:

```
<Services>
  <JMXManagementService id="JMXManagementService"/>
</Services>
```

This assumes that the Catalys Local Management Agent (LMA) is also running. To access the configuration MBeans without running the LMA, configure the `JMXManagementService` to register its configuration MBeans with the platform MBean Server:

```
<Services>
  <JMXManagementService id="JMXManagementService"
    registerConfigurationMBeansOnActivation="true"/>
</Services>
```

```
</Services>
```

To install and run the LMA, please refer to the LMA documentation.

The Node's MBeans can be browsed using the JConsole application.

## Runtime Changes vs. Runtime Reload

The general pattern for accessing the model via JMX includes two steps. The first step is to make the change to the configuration by modifying the desired configuration MBean and the second is to execute one of the `reload` operations. There are two levels of reload operations available via JMX:

- Application reload: used to reload changes to an entire application
- Session reload: used when the changes are limited to a particular session

## Preparing for Reload

As described above, a session must be **reloadable** before its configuration can be loaded into runtime. A session is reloadable if it is disconnected, and locked or blocked, and its session manager is de-activated.

To enable a session for reload, execute the `makeSessionReloadable` operation on the `Configuration.Application.Session` MBean, which will disconnect, block the session and de-activate the session manager.

If the session shares active runtime components with another session, the above operation will fail, with a message such as:

```
ERROR jmx.ConfigurationMBean: Exception thrown from 'makeSessionReloadable' operation
invoked with parameters []
com.camerontec.catalys.server.configuration.IllegalOperationException:
Cannot enable sessions for reloadComponent
/CatalysServer/Application/Sessions/Session(MONYPNY_IINVST)/SessionManager/
ListenerMessageProcessors/MessageProcessor(mp_1) is active and shared
by a non-reloadable session
```

In this case, both sessions need to be enabled for reload. To do this, execute the `makeSessionReloadable` operation on the `Configuration.Application` MBean, supplying the session identifiers of both sessions in a comma-separated list.

The status of the session can be determined by executing the `diagnosis` operation on the `Configuration.Application.Session` MBean for the particular session of interest. The result of this operation includes whether the session is reloadable, and whether it is in/active, un/locked, un/blocked, and dis/connected.

## Reload on a Single Session

To reload the changes to a single session, make the changes to the desired MBeans, and then execute the `reload` operation of the particular `Configuration.Application.Session` MBean.

## Reload on a Whole Node Application

To reload changes to a whole application, make the changes to the desired MBeans, and then execute the `reload` operation on the particular `Configuration.Application` MBean.

## Re-activating Sessions After a Reload

To re-activate a single session, after a reload, execute the `unblockAndActivate` operation on the `Configuration.Application.Session` MBean.

## Backup and Rollback

Each time the configuration is loaded into the runtime model via JMX, the current configuration is backed up into a timestamped XML file. The naming convention of these backed-up configuration files is `<config_file_name>.backup_yyyyMMddHHmmss.xml` where:

- 'yyyy' is the year
- 'MM' is the month
- 'dd' is the day of the month
- 'HH' is the hour
- 'mm' is the minutes
- 'ss' is the seconds

To restore a particular configuration from XML, the known good configuration can be copied into the master configuration file (the one specified on the command-line which started the Node) and a reload can be triggered from the command-line by issuing the [config\\_reload](#) command.

To rollback the set of configuration changes that have taken place since the last JMX reload, execute the `undo` operation on the relevant `Configuration.Application` MBean. To undo changes on a particular session, execute the `undo` operation on the relevant `Configuration.Application.Session` MBean, supplying parameters for fine-tuning the `undo` operation:

- `undoMarkForDelete`: this will undo any `markForDelete` operations that have been executed on the session, since the last reload.
- `undoAdd`: this will undo any `add` operations that have been executed on the session, since the last reload. For example, `addConnection`, `addMessageProcessor`, `addSessionEncryption` and `addSessionPersistence`.
- `undoAll`: this will undo all other modifications to the session since the last reload, specifically all attribute, property and macro changes.

### 4.3.3.2. Changing and Reloading the Model via XML

The configuration model can be accessed via XML by editing the XML configuration file and reloading it into runtime using one of the [config\\_reload](#), [s\\_reload](#), or [s\\_force\\_reload](#) CLI commands.

## Preparing for Reload

As described above, a session must be **reloadable** before its configuration can be loaded into runtime. A session is reloadable if it is disconnected, and locked or blocked, and its session manager is de-activated.

To disconnect and block the session, and de-activate the session manager, use the `s_make_reloadable` command, which will also check for active runtime components that are shared with other non-reloadable sessions, and fail if it finds any. This command must be supplied with the hash value of the session, which can be retrieved by running the `s_list` command, lists all sessions, associating each party/counterparty pair with a hash value. For example:

```
> s_list
s_list
# TargetCompId TargetSubId TargetLocation SenderCompId SenderSubId SenderLocation
--
15 IINVST MONYPNY

> s_make_reloadable #15
INFO SessionManager.IINVST MONYPNY: inactive
sessionId Problems
-----
```

If the session is not reloadable when the `config_reload` command is issued then the command will fail, and an error message will be printed to the console.

```
> config_reload
INFO xmlconverter.XMLConfigurationReader: Reading configuration from XML file ../config.xml
FAILED Cannot reload XML configuration from file: Cannot change the attributes of
a non-reloadable session
/CatalysServer/Application/Sessions/Session(MONYPNY_IINVST)
```

To re-activate the session, use the `s_unblock_and_activate` command.

Individual commands can also be used to disconnect and block the session, and de-activate the session manager. To disconnect a session, use the `s_disconnect` command. To block a session, use the `s_block` command. To lock a session, use the `s_lock` command. Note that blocking and locking a session are functionally equivalent (the session ignores new connections) but the blocked status is persisted (survives an application restart), whereas the locked status is not. To de-activate a session manager, use the `c_de_activate` command, supplying the component name of the session manager, which can be determined using the `c_list` command.

## Reloading Without Changes

If there is no difference between the configuration in the XML file and what is currently running on the server, there will be no effect on the server, and the following message will be printed on the console:

```
> config_reload
```

```
INFO xmlconverter.XMLConfigurationReader: Reading configuration from XML file ../config.xml
INFO configuration.ConfigurationReloader: Old and new configurations are identical. \
Nothing to reload.
```

### Non-structural Changes

If there are changes to a reloadable session then these changes will be accepted and loaded into the runtime model. Non-structural changes (modifying the values of attributes) will produce a message such as the following:

```
> config_reload
INFO xmlconverter.XMLConfigurationReader: Reading configuration from XML file ../config.xml
INFO configuration.ConfigurationReloader: Reloading configuration using reader ../config.xml
The following session(s) have changed, but with no structural differences, \
they are to be reloaded:
Session: /CatalysServer/Application/Sessions/Session(MONYPNY_IINVST)
After reload the following session(s) will exist:
Session: /CatalysServer/Application/Sessions/Session(MONYPNY_IINVST)
```

### Structural Changes

Structural changes consist of modifying the hierarchy or order of elements within the configuration. A structural change will produce console output such as:

```
> config_reload
INFO xmlconverter.XMLConfigurationReader: Reading configuration from XML file ../config.xml
INFO configuration.ConfigurationReloader: Reloading configuration using reader ../config.xml
The following session(s) have changed, with structural differences, they are to be reloaded:
Session: /CatalysServer/Application/Sessions/Session(MONYPNY_IINVST)
After reload the following session(s) will exist:
Session: /CatalysServer/Application/Sessions/Session(MONYPNY_IINVST)
```

### Changes to Application-wide Components

If there are configuration changes to components which exist outside of sessions (links and services, for example) the application can still be reloaded without its sessions being reloadable. In this case a console message will be printed which indicates that the sessions have not been reloaded, but the modified application-wide components have.

```
> config_reload
INFO xmlconverter.XMLConfigurationReader: Reading configuration from XML file ../config.xml
INFO configuration.ConfigurationReloader: Reloading configuration using reader ../config.xml
The following identical session(s) are to be skipped (not reloaded) :
Session: /CatalysServer/Application/Sessions/Session(MONYPNY_IINVST)
After reload the following session(s) will exist:
Session: /CatalysServer/Application/Sessions/Session(MONYPNY_IINVST)
The following new service(s) are to be added to old configuration:
Service: /CatalysServer/Application/Services/Service(serv)
```

## Session-only Reload

Sessions can also be reloaded on an individual basis using the [s\\_reload](#) command. For this command to succeed the session must be reloadable.

The [s\\_force\\_reload](#) command can be used to enable the session for reload, reload it, and restore it to its previous state.

### 4.3.3.3. Dynamic Configuration in a High Availability Environment

Dynamic configuration is only available on the primary node of a High Availability cluster. Changes made and reloaded on the primary are automatically synchronized to any running secondary nodes. If any secondary node of the cluster is not running at the time the changes are reloaded, then the changes need to be made manually on the secondary before starting it.



# Chapter 5. Standard Features

## Table of Contents

5.1. Session Persistence .....	87
5.1.1. Introduction .....	87
5.1.2. JournalingPersister .....	88
5.1.3. MemoryPersister .....	90
5.1.4. MemorySeqNoOnlyPersister .....	91
5.2. Scheduling .....	91
5.2.1. Introduction .....	91
5.2.2. Configuration Overview .....	91
5.2.3. Configuration Examples .....	95
5.2.4. Scheduling and Daylight Savings Time .....	97
5.2.5. In Schedule vs. Outside Schedule .....	98
5.2.6. Custom Tasks .....	98
5.2.7. JMX Management .....	99
5.3. FIX Routing .....	102
5.3.1. Introduction .....	102
5.3.2. Routing Modes .....	102
5.3.3. Making Routing Decisions .....	104
5.3.4. Routing Side .....	105
5.3.5. Routing Networks .....	106
5.3.6. Handling Routing Issues .....	106
5.3.7. FIX Routing Configuration .....	107
5.4. FIX Drop Copy .....	109
5.4.1. Introduction .....	109
5.4.2. Drop Copy Methods .....	110
5.5. Message Buffering .....	110
5.5.1. Introduction .....	110
5.5.2. Configuration .....	110
5.5.3. Persistent Collection Configuration .....	112
5.5.4. Managing Buffers from the Command Line .....	112
5.6. Message Throttling .....	113
5.6.1. Introduction .....	113
5.6.2. Configuration .....	113
5.7. FIX Dictionaries .....	115
5.7.1. Introduction .....	115
5.7.2. Getting Started .....	115
5.7.3. Custom Dictionaries .....	116
5.7.4. Configuration of Custom Dictionary .....	116

5.7.5. Dictionaries API .....	116
5.7.6. Non XML-based Dictionaries .....	116
5.8. Inbound Message Validation .....	117
5.8.1. Introduction .....	117
5.8.2. Validation Rules .....	117
5.9. Message Factories .....	122
5.9.1. Introduction .....	122
5.9.2. Configuration .....	122
5.9.3. Provided Implementations .....	123
5.9.4. Custom Implementation .....	124

## 5.1. Session Persistence

### 5.1.1. Introduction

In order to maintain continuity of FIX sessions between application restarts, FIX session state must be persisted to disk. If session state is not persisted then the application's in-memory state is lost after a restart, and manual intervention is required at the counterparty to reestablish the session.

The FIX session state consists of the following elements, which are stored in separate directories on the filesystem. The name of each directory is suffixed with an identifier, which is indicated below in parentheses.

- All outbound messages sent on the session and the next outbound sequence number (*-outMsg*)
- The next inbound sequence number expected (*-inSeqNum*)
- Session attributes: the last reset time, session state and the logged-on status (*-atts*)
- Optional: All inbound messages received on the session (*-inMsg*)

Persisters are configured within each [Session](#) element in the Node's configuration. Classes which implement the [IFIXSessionStatePersister](#) interface can be configured as persisters.



#### Note

The `JournalingPersister` is the default file-based persister provided with the Node. This persister should be used in all production environments. Memory-based persisters should only be used under unique circumstances.

Two memory-based implementations, [MemoryPersister](#) and [MemorySeqNoOnlyPersister](#), are provided for test purposes only. Note that `MemorySeqNoOnlyPersister` will be chosen as the default persister if no other persister has been configured.

For a demonstration of the `JournalingPersister`, please see the template in *templates/FilePersistence*.

## 5.1.2. JournalingPersister

The `JournalingPersister` stores each FIX session state element in a directory based on the `baseDirectoryName` attribute.

If you use the installer and select the 'Node Instances' option, persistence data will be created in the `<CATALYS_NODE_INSTANCES>/example/data` directory.



### Important

The `baseDirectoryName` attribute is treated as a directory name prefix. Suffixes for each data type are added automatically. For example, if the attribute's value is `/data/CLIENT1`, the effective directory names will be:

- `/data/CLIENT1-outMsg/`
- `/data/CLIENT1-inSeqNum/`
- `/data/CLIENT1-atts/`

Therefore the value of this attribute must not end with a backslash.

By default, the message elements (*outMsg* and *inMsg*) are stored in a high-capacity indexed list. Two files will be created for each element: journal files (*journal.\*.dat*) and index files (*index.dat*).

If the `JournalingPersister` is configured for a session that is part of a High Availability environment, then its persistence data will automatically be replicated across the cluster.

## Configuration

The minimum configuration for the `JournalingPersister` is as follows:

```
<Persister>
  <JournalingPersister id="jpl" baseDirectoryName="../../data/persist/CLIENT1" />
</Persister>
```

For a complete list of attributes, see the [JournalingPersister Configuration Reference](#).

## Persistent Collections Properties

The underlying persistence store of the `JournalingPersister` is configured using [Persistent Collection Properties](#), which are specified via a nested `Properties` element.

## Standard Features

As described above, the `JournalingPersister` creates a different directory for output messages, input sequence number, state attributes and (if `persistInputMessages` is set to true) input messages. Each directory generated by the `JournalingPersister` is backed by a different collection, and as such has its own set of collection properties. There is however only one set of properties for the `JournalingPersister`, passed in via a nested `Properties` element.

The name of each property can be prefixed by a specific element type (e.g. `outMsg-`, `atts-`, `inSeqNum-`, `inMsg-`) which applies only to that collection, or the property can be supplied in its generic form (without prefixes) which applies to all collections.

For example, to set the [filePathLevelsInLogger](#) attribute for all persistence files:

```
<Persister>
  <JournalingPersister id="jpl" baseDirectoryName="../data/persist/CLIENT1">
    <Properties id="jpl-props">
      <Property name="filePathLevelsInLogger" value="3"/>
    </Properties>
  </JournalingPersister>
</Persister>
```

To set the [rollJournalFileEvery](#) attribute to a different value for each file, the configuration would be:

```
<Persister>
  <JournalingPersister id="jpl" baseDirectoryName="../data/persist/CLIENT1">
    <Properties id="jpl-props">
      <Property name="outMsg-rollJournalFileEvery" value="1073741824"/>
      <Property name="atts-rollJournalFileEvery" value="1024"/>
      <Property name="inSeqNum-rollJournalFileEvery" value="2048"/>
    </Properties>
  </JournalingPersister>
</Persister>
```

There are some default attributes for the `JournalingPersister` that are different from the standard Persistent Collections defaults:

Attribute	Default
<a href="#">directoryName</a>	This attribute is ignored. The <code>baseDirectoryName</code> of the <code>JournalingPersister</code> element is used instead.
<a href="#">name</a>	Defaults to <code>baseDirectoryName + element type</code> (e.g. <code>-outMsg</code> ). If this attribute is supplied, then it should not be used in its generic form, rather it must be explicitly prefixed by the file descriptor. For example: <div>&lt;Persister&gt;</div>

## Standard Features

Attribute	Default
	<pre>&lt;JournalingPersister   id="jpl"   baseDirectoryName="../data/persist/CLIENT1"&gt;    &lt;Properties id="jpl-props"&gt;     &lt;Property name="outMsg-name" value="outMsg"/&gt;     &lt;Property name="atts-name" value="atts"/&gt;     &lt;Property name="inSeqNum-name" value="inSeqNum"/&gt;   &lt;/Properties&gt; &lt;/JournalingPersister&gt; &lt;/Persister&gt;</pre>
<a href="#">isHighCapacityList</a>	Defaults to <code>true</code> for the <i>outMsg</i> and <i>inMsg</i> collections, and defaults to <code>false</code> for the <i>inSeqNum</i> and <i>atts</i> collections.
<a href="#">versionHistorySize</a>	Defaults to 100,000 instead of 10,000 for the <i>outMsg</i> and <i>inMsg</i> collections.

## Maintenance

To ensure that the persistence files do not grow without bound, the files associated with the `JournalingPersister` need to be compacted periodically. This is performed automatically by the Node when a [session reset](#) occurs.

Alternatively the `CompactTask` can be [scheduled](#). This task takes a colon-delimited set of names as a property. These names identify the collections to compact. In the case of the `JournalingPersister`, there is one name for each type of collection for each session: `-outMsg`, `-atts`, `-inSeqNum` (and optionally `-inMsg`), and these names are prefixed by the value of `baseCollectionName` if it is specified, or `baseDirectoryName` if it is not. For example (in the case where `baseCollectionName` is not specified and `baseDirectoryName` is set to `persist/Session1`):

```
<Task id="compact" class="com.camerontec.catalys.server.task.CompactTask">
  <Properties id="props1">
    <Property name="names" value="persist/Session1-outMsg:persist/Session1-atts"/>
  </Properties>
</Task>
```

### 5.1.3. MemoryPersister

The `MemoryPersister` should only be used for testing purposes and never in a production environment. It stores each FIX session state element in memory, which will be lost if the Node is restarted.



### Caution

If the `MemoryPersister` runs indefinitely, it will consume all of the JVM's allotted memory and cause the Node to crash.

The configuration for the `MemoryPersister` is as follows:

```
<Persister>
  <MemoryPersister id="mpl" />
</Persister>
```

## 5.1.4. MemorySeqNoOnlyPersister

The `MemorySeqNoOnlyPersister` should only be used for testing purposes and never in a production environment. It only stores inbound and outbound sequence number in memory, which will be lost if the Node is restarted. This is the default session persister if no other persister is configured.

## 5.2. Scheduling

### 5.2.1. Introduction

The Node Scheduler is used to automatically start and stop FIX sessions, reset FIX sequence numbers, and perform any other task. It supports repeating events, multiple time zones and excluded days. Scheduled tasks can be applied at the session level and globally at the application level.

### 5.2.2. Configuration Overview

The following is the general structure of the Scheduler's configuration:

```
<Config>
  <Application id="buy">
    <SchedulingDefinitions id="sched">
      <Tasks>
        <Task id="start" class="com.camerontec.catalys.server.task.StartTask"/>
        ...
      </Tasks>
      <Schedules>
        <Schedule id="start_schedule">
          <Include dayOfWeek="2-6" hour="11" minute="15"/>
        </Schedule>
        ...
      </Schedules>
    </SchedulingDefinitions>
  </Application>
</Config>
```

```

<EventLists>
  <EventList id="session_eventList">
    <Event id="start_event" task="start" schedule="start_schedule"/>
    ...
  </EventList>
</EventLists>
</SchedulingDefinitions>
</Application>
</Config>

```

Each of the configuration elements is explained in detail below.

## SchedulingDefinitions

All scheduling entries are grouped together under this element. There can only be one of these. For a complete list of its attributes, see the [SchedulingDefinitions Configuration Reference](#).

This element can contain zero or one of each of the following elements, in any order: `Tasks`, `Schedules` and `EventLists`.

## Tasks


This element is a container for one or more `Task` elements.

## Task

A `Task` executes its functionality at the date and time specified by its associated schedule. For a complete list of its attributes, see the [Task Configuration Reference](#).

A number of standard tasks classes are provided. The most commonly used are:

Task	Description
StartTask	Attempt to connect a session and logon with the FIX counterparty.
UnblockAndStartTask	Unblock a session and attempt to connect a session and logon with a FIX counterparty.
StopTask	Logout with the FIX counterparty and disconnect session.
BlockAndStopTask	Block session and then logout with FIX counterparty and disconnect session.
ResetTask	Reset the sequence numbers of a FIX session to 1.
ShutdownTask	Shut down the Node process.
EndOfDayTask	Perform the end-of-day procedure on a session and all of the components in its message processing chain. For standard Node components, this usually

Task	Description
	<p>involves compacting and purging any files that are written to disk to prevent them growing without bound. This task can be configured with a nested <code>Properties</code> element specifying an optional parameter, which is passed to <a href="#">performEndOfDayProcedure</a>.</p> <p>For example, those components using Market Mirror storage pass this value in reference to the local <code>endOfDayClearingCondition</code> attribute. This parameter is only necessary if the default or locally configured value of the attribute is inadequate.</p> <pre data-bbox="542 657 1235 827">&lt;Task id="eod"   class="com.camerontec.catalys.server.task.EndOfDayTask"&gt;   &lt;Properties id="props"&gt;     &lt;Property name="parameter" value="OrderID=535364758"/&gt;   &lt;/Properties&gt; &lt;/Task&gt;</pre> <p>Custom components can interpret this parameter as required.</p>
CompactTask	<p>Compact a persistence file on disk.</p> <p>This task is configured by providing the name(s) of the collection(s) to compact, as a property. For example:</p> <pre data-bbox="542 1157 1224 1327">&lt;Task id="compact"   class="com.camerontec.catalys.server.task.CompactTask"&gt;   &lt;Properties id="props"&gt;     &lt;Property name="names" value="data/dir1:data/dir2"/&gt;   &lt;/Properties&gt; &lt;/Task&gt;</pre> <div data-bbox="565 1423 1369 1686">  <p><b>Note</b></p> <p>While compacting a collection is thread-safe, it can be a time consuming and resource intensive process and should be performed only while a session is inactive.</p> </div>

## Schedules

This element is a container for one or more `Schedule` elements.



## Schedule

This element and its child elements define the times at which events occur. For a complete list of its attributes, see the [Schedule Configuration Reference](#).

It can contain zero or more `Include` and `Exclude` elements, in any order.

## Include/Exclude

These elements represent a time/date or range of times/dates to be either included in or excluded from a schedule.

Ranges and wildcards are accepted similar to the Unix cron command. In other words, use a comma between multiple values and a hyphen to indicate a range. If a value is not specified, it is assumed that it can take "any" value in the same way that \* is used in cron.

Attribute	Acceptable Values
hour	0-23
minute	0-59
dayOfWeek	1-7 (Sunday to Saturday)
dayOfMonth	1-31
month	1-12
year	Four digits

Note that the time zone is not specified here. The time zone is taken from the enclosing `Schedule` element.

## EventLists

This element is a container for one or more `EventList` elements.

## EventList

This element can contain one or more `Event` elements. For a complete list of its attributes, see the [EventList Configuration Reference](#).

## Event Element

An `Event` associates a `Task` with a `Schedule`. Events are what the scheduler actually schedules. Sometimes events are referred to as "scheduled tasks". For a complete list of `Event` attributes, see the [Event Configuration Reference](#).

## 5.2.3. Configuration Examples

The following sections show two Node configuration scheduling elements. These are given in XML format but can be similarly configured via the Dashboard or JMX operations.

### Simple Example

This example demonstrates a start, stop and reset schedule for a session.

```
<Config>
  <Application id="buy">
    <SchedulingDefinitions id="sched">
      <Tasks>
        <Task id="start" class="com.camerontec.catalys.server.task.StartTask"/>
        <Task id="stop" class="com.camerontec.catalys.server.task.StopTask"/>
        <Task id="reset" class="com.camerontec.catalys.server.task.ResetTask"/>
      </Tasks>
      <Schedules>
        <Schedule id="start_schedule">
          <Include dayOfWeek="2-6" hour="11" minute="15"/>
        </Schedule>
        <Schedule id="stop_schedule">
          <Include dayOfWeek="2-6" hour="15" minute="55"/>
        </Schedule>
        <Schedule id="reset_schedule">
          <Include dayOfWeek="2-6" hour="15" minute="57"/>
        </Schedule>
      </Schedules>
      <EventLists>
        <EventList id="session_eventList">
          <Event id="start_event" task="start" schedule="start_schedule"/>
          <Event id="stop_event" task="stop" schedule="stop_schedule"/>
          <Event id="reset_event" task="reset" schedule="reset_schedule"/>
        </EventList>
      </EventLists>
    </SchedulingDefinitions>
    <Sessions>
      <Session fixversion="4.2" heartbeat="60" counterpartycompid="CLIENT1" compid="BROKER"
        scheduledEvents="session_eventList">
        <Connections>
          <SocketConnection hostname="localhost" port="2000" id="sc"/>
        </Connections>
        <SessionManager>
          <SourceMessageProcessors/>
          <ListenerMessageProcessors/>
        </SessionManager>
      </Session>
    </Sessions>
  </Application>
</Config>
```



## Note

Application and Session elements refer to an EventList via the scheduledEvents attribute.

## More Complex Example

In this example, common holidays are defined which are used by the start, stop and reset schedules (via the parentSchedule attribute). An application shutdown schedule is also demonstrated.

```
<Config>
  <Application id="buy_with_holidays" scheduledEvents="application_eventList">
    <SchedulingDefinitions id="schedule">
      <Tasks>
        <Task id="start" class="com.camerontec.catalys.server.task.StartTask"/>
        <Task id="stop" class="com.camerontec.catalys.server.task.StopTask"/>
        <Task id="reset" class="com.camerontec.catalys.server.task.ResetTask"/>
        <Task id="shutdown" class="com.camerontec.catalys.server.task.ShutdownTask"/>
      </Tasks>
      <Schedules>
        <Schedule id="holidays">
          <Exclude dayOfMonth="4" month="7"/>
          <Exclude dayOfMonth="25-26" month="12"/>
          <Exclude dayOfMonth="1" month="1"/>
          <Exclude dayOfMonth="24" month="10" year="2002"/>
        </Schedule>
        <Schedule id="start_schedule" parentSchedule="holidays">
          <Include dayOfWeek="2-7" hour="10"/>
        </Schedule>
        <Schedule id="stop_schedule" parentSchedule="holidays">
          <Include dayOfWeek="2-6" hour="17" minute="00"/>
          <Include dayOfWeek="7" hour="14"/>
        </Schedule>
        <Schedule id="reset_schedule" parentSchedule="holidays">
          <Include dayOfWeek="2-6" hour="18" minute="0"/>
          <Include dayOfWeek="7" hour="15" minute="0"/>
        </Schedule>
        <Schedule id="shutdown_schedule">
          <Include hour="20" minute="0"/>
        </Schedule>
      </Schedules>
      <EventLists>
        <EventList id="session_eventList">
          <Event id="start_event" task="start" schedule="start_schedule"/>
          <Event id="stop_event" task="stop" schedule="stop_schedule"/>
          <Event id="reset_event" task="reset" schedule="reset_schedule"/>
        </EventList>
        <EventList id="application_eventList">
          <Event id="shutdown_event" task="shutdown" schedule="shutdown_schedule"/>
        </EventList>
      </EventLists>
    </SchedulingDefinitions>
  </Application>
</Config>
```

```
</EventList>
</EventLists>
</SchedulingDefinitions>
<Sessions>
  <Session fixversion="4.2" heartbeat="60" counterpartycompid="CLIENT1" compid="BROKER"
    scheduledEvents="session_eventList">
    <Connections>
      <SocketConnection hostname="localhost" port="2000" id="sc"/>
    </Connections>
    <SessionManager>
      <SourceMessageProcessors/>
      <ListenerMessageProcessors/>
    </SessionManager>
  </Session>
</Sessions>
</Application>
</Config>
```



### Caution

Multiple tasks should not be scheduled to run at the same day and time on a session. This can cause improper FIX session behavior and data corruption issues.

## 5.2.4. Scheduling and Daylight Savings Time

Time changes due to daylight savings time (DST) can affect the order of task execution. For example, in North America, there are two transitions that require special attention:

- Spring forward: Tasks scheduled during the lost hour (02:00:00 and 02:59:59) will be executed between 03:00:00 and 03:59:59.
- Fall back: Tasks scheduled between 01:00:00 and 01:59:59 are executed only once, during the second occurrence of the repeated hour.



### Important

We recommend to avoid scheduling tasks during these times so that tasks execute consistently year round.

## 5.2.5. In Schedule vs. Outside Schedule

Normally a permanent connection with a FIX counterparty is not maintained. Instead, the session connects at a certain time and disconnects some time later. In other words, there is a period of time when the session is "in schedule" and for the rest of the time, the session is "outside schedule".

It is important to be able to determine whether a session is in or outside schedule because that will affect how errors are reported and how the Node reacts if a connection is lost. For example, if a connection is lost when it is scheduled to be connected, then that is an error that will be reported. Also, the Node will automatically try to re-establish the connection. On the other hand, losing a connection outside of the scheduled connection time is not an error and the Node will not try to reconnect.

The Node determines whether a session is in schedule or outside schedule by looking at the scheduling of all standard `StartTask` and `StopTask` elements. If the `StopTask` is chronologically next in the schedule, then it is assumed that the session should already be connected and is "in schedule". If the `StartTask` is next, then the session is "outside schedule".

Sometimes you may want to stop a certain session connection and prevent the Node from automatically retrying to connect. You could do this by selecting the `StopTask` from the Dashboard or JMX API for that session and executing the `doTaskNow` operation with the `disableNextEvent` parameter set to `true`. This will execute the `StopTask` immediately, thereby terminating the connection for that session. It will also disable the next scheduled `StopTask`. This means that the following `StartTask` will be seen as the next to be executed, so the connection will be considered "outside schedule" and no reconnection attempt will be made. When you subsequently want to restart, simply execute the `StartTask`. You may also want to re-enable the next scheduled `StopTask` so that the connection will perform its normal scheduled stop automatically.

Similarly, if you want to manually start a connection outside its normal schedule, select the `StartTask` for that session and execute `doTaskNow` with `disableNextEvent` set to `true`.

## 5.2.6. Custom Tasks

In addition to the previously described built-in tasks, custom tasks may also be created and scheduled. A custom task should inherit from either [TaskBase](#) or [PartyTaskBase](#) if it is a session-based task. Full source code for the tasks is included with the Node distribution in `templates/src/com/camerontec/catalys/server/task`.

See the custom [Task](#) example in the Programming Guide.

## 5.2.7. JMX Management

### Configuration

The Scheduler can be configured via JMX while the server is running. There are add operations for `Scheduling`, `Task`, `Schedule`, `EventList` and `Event`. Once these elements have been added via JMX their attributes can be modified as necessary. These elements can also be marked for delete.

Once all scheduling elements have been added and configured appropriately, the `reload` operation can be executed on the application or session, and the configuration will be loaded into runtime.

### Monitoring

The Scheduler can be monitored and controlled by any JMX compliant management tool (JConsole, a JMX-compliant GUI tool that connects to a running JVM, for example).

### Scheduling MBeans

The MBeans that we are interested in are the *scheduled tasks*. These tasks are grouped by the session with which the tasks are associated. All scheduled tasks for a given session are grouped together under a heading of the form `<app-id>.<counter-party>.<local-party>.ScheduledTasks`. The application ID is the `id` attribute of the `Application` element in the configuration. The counter party name is constructed from the session's `counterpartycompid`, `counterpartysubid` and `counterpartylocationid` attributes. The local party name is constructed from the session's `compid`, `subid` and `locationid` attributes. The following example shows that the session `buy.CLIENT.SERVER` (i.e. the session whose `counterpartycompid="CLIENT"` and `compid="SERVER"` running in the application whose `id="buy"`) has three scheduled tasks: a start task, a stop task and a reset task.

```
buy.CLIENT.SERVER.ScheduledTasks

task=com.camerontec.catalys.server.task.ResetTask,
id=buy.CLIENT.SERVER.reset,
Sessions=com.camerontec.catalys.server.configuration.ConnectionPointConfigInMemory@65d0d124,
schedule=reset

task=com.camerontec.catalys.server.task.StartTask,
id=buy.CLIENT.SERVER.start,
Sessions=com.camerontec.catalys.server.configuration.ConnectionPointConfigInMemory@65d0d124,
schedule=start

task=com.camerontec.catalys.server.task.StopTask,
id=buy.CLIENT.SERVER.stop,
Sessions=com.camerontec.catalys.server.configuration.ConnectionPointConfigInMemory@65d0d124,
schedule=stop
```

Some tasks are not associated with any session, for example a task to shut down the Node. These tasks are also grouped together. The heading for this group is simply *ScheduledTasks*. For example:

## Standard Features

### ScheduledTasks

```
task=com.camerontec.catalys.server.task.ShutdownTask,
id=shutdown,
Sessions=com.camerontec.catalys.server.configuration.ConnectionPointConfigInMemory@65d0d124,
Application=buy on DELL/192.168.0.2,
schedule=shutdown
```

The tasks described above are "scheduled tasks". Each scheduled task consists of a task linked to a schedule. For example, each FIX session will have a scheduled task associated with the start task and a schedule such as "each weekday morning at 9am". As well as displaying all scheduled tasks, the summary display also displays each plain task. All tasks that appear in the lists of scheduled tasks will also appear in the list of tasks. Note, however, that not all tasks have to be scheduled. It might be useful to have a task that does not automatically execute at a certain time but which is only ever executed manually. The [Task MBean section](#) describes how you can execute a task manually.

Like scheduled tasks, tasks are grouped together. For example:

### buy.CLIENT.SERVER.Tasks

```
task=com.camerontec.catalys.server.task.ResetTask,
id=buy.FCLIENT.SERVER.reset,
Sessions=com.camerontec.catalys.server.configuration.ConnectionPointConfigInMemory@65d0d124

task=com.camerontec.catalys.server.task.StartTask,
id=buy.CLIENT.SERVER.start,
Sessions=com.camerontec.catalys.server.configuration.ConnectionPointConfigInMemory@65d0d124

task=com.camerontec.catalys.server.task.StopTask,
id=buy.CLIENT.SERVER.stop,
Sessions=com.camerontec.catalys.server.configuration.ConnectionPointConfigInMemory@65d0d124
```

and

### Tasks

```
task=com.camerontec.catalys.server.task.ShutdownTask,
id=shutdown,
Sessions=com.camerontec.catalys.server.configuration.ConnectionPointConfigInMemory@65d0d124,
Application=buy on DELL/192.168.0.2
```

You can click on any task line to see details of that particular task or scheduled task.

## The Task MBean

The Task MBean has two operations and no attributes:

Operation	Description	Parameters
run	Execute this task immediately.	None
scheduleAt	Schedule this task to run once at a given time (using the default time zone).	<p>hour: the hour (0-23) at which to schedule the task</p> <p>minute: the minute (0-59) at which to schedule the task</p>



### Note

When a task is executed a screen will always display a message indicating that the task ran successfully. In fact this may not be the case. All the message is really saying is that the task executed. It does not guarantee that the task achieved what it was intended to do. If a task has a problem it will typically log an error message or throw an exception. This will be reported in the standard logging and monitoring of the Node.

## Scheduled Task MBean

The Scheduled Task MBean has the following attributes:

Attribute	Description
nextEventEnabled	This attribute displays whether or not the next scheduled execution of this task is enabled. Normally it will be enabled. However, it is possible to disable it. In that case the task will not be executed at that time.
nextEvents	<p>This attribute holds the next few scheduled times at which the task is due to be executed.</p> <p>If an event has been disabled, the displayed time will be followed by "(disabled)".</p>

The Scheduled Task MBean has the following operations:

Operation	Description	Parameters
doTaskNow	Executes this task immediately.	disableNextEvent: if set to true, the next scheduled



Operation	Description	Parameters
	Depending on its <code>disableNextEvent</code> parameter, you can also automatically disable the next scheduled execution of this task. (This is equivalent to setting the <code>nextEventEnabled</code> attribute).	event will be disabled, i.e. the task is executed immediately instead of at its scheduled time. If this parameter is set to false then the task is executed immediately as well as at its scheduled time.
<code>setNextEventEnabled</code>	Enable or disabled the next scheduled event.	<code>enabled</code> : whether to enable or disable the next event.
<code>isNextEventEnabled</code>	Query whether the next event is enabled.	None
<code>getNextEventsAsStrings</code>	Get a list of the next few scheduled events.	None

## 5.3. FIX Routing

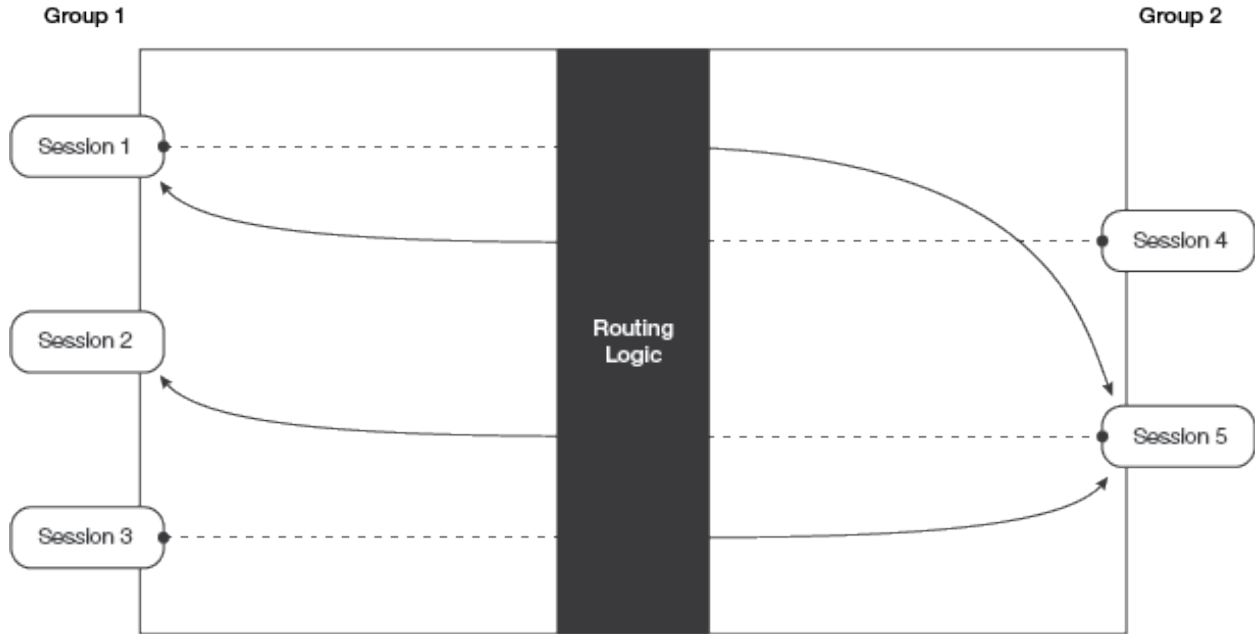
### 5.3.1. Introduction

The Node can be configured as a FIX message router. In this setup, the Node receives FIX messages on one or more sessions and sends them out a different session. There are a number of routing configuration options and several methods by which to make routing decisions and apply them to a given message.

### 5.3.2. Routing Modes

#### Session-Group Routing

This is the most common routing mode. Sessions are placed in two separate groups, and messages arriving on a session in one group can only be routed to a session in the other group and vice versa.

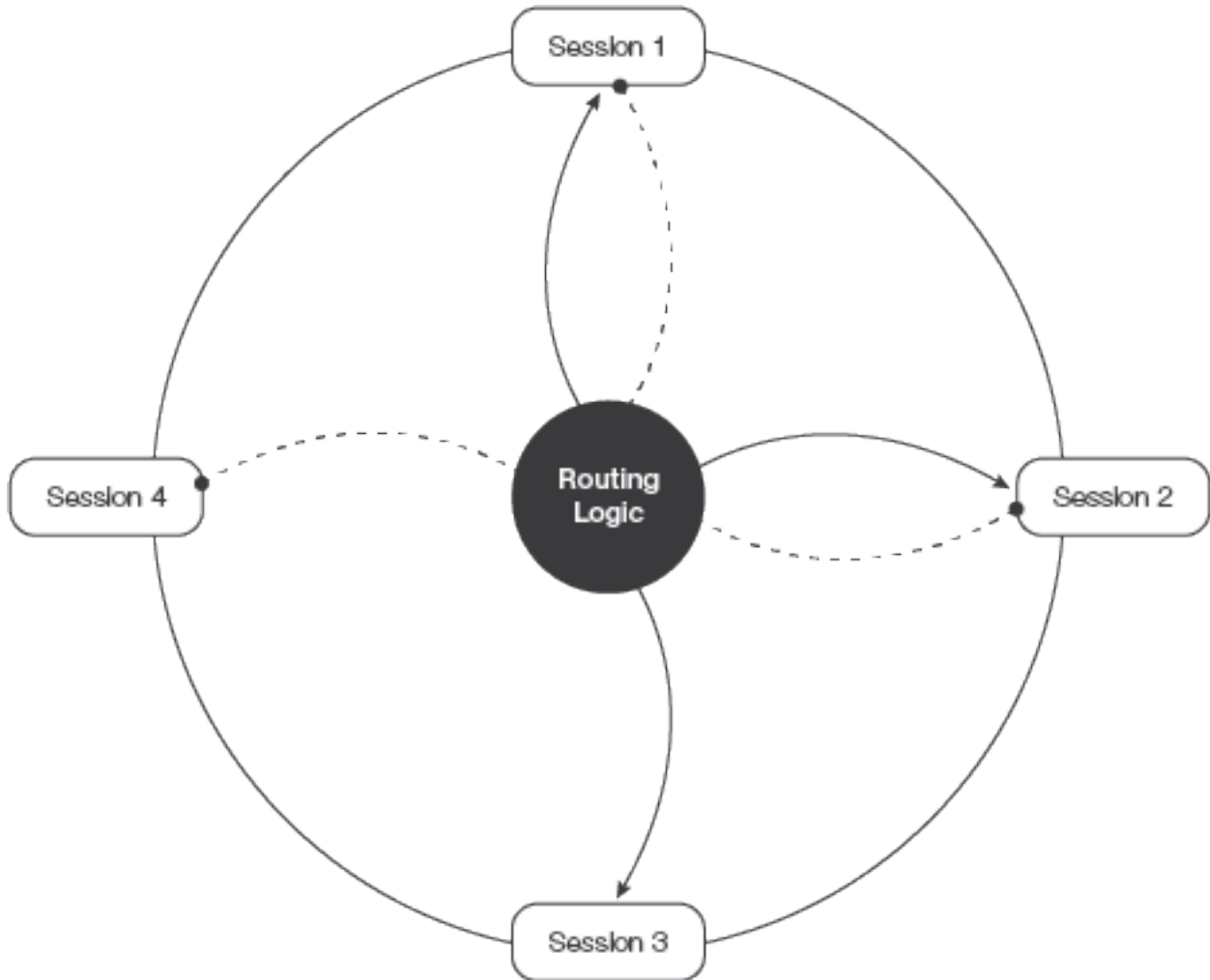


This is a typical setup for brokers. They place client sessions in one group and exchange and execution venue sessions in the other group. Client sessions receive order messages (orders, replaces, cancels, etc.) which route to exchanges. Exchange sessions receive trade messages (acknowledgements, fills, rejects, etc) which then route back to the client sessions.

Another common use case for this setup is to consolidate and normalize inbound messages from many sessions into one outbound session.

## Routing Hub

The Node can also be configured as a routing hub, where sessions are not placed into separate groups. Rather, a message from one session can be routed to any other session including itself. This is a more flexible configuration.



### 5.3.3. Making Routing Decisions

Regardless of which routing mode is chosen, something must decide which target session each inbound message is routed to. This can be handled by the Node itself or delegated to user-provided rules and/or custom components.

#### Rules Engine

The [Catalys Rules Engine](#) has built-in support for FIX routing. Routing rules of varying complexity can be created to meet your requirements by taking advantage of the built-in conditions and expression language. Primary and backup routing destinations are also supported when using Rules Engine routing.

The Rules Engine supports "sticky" routing and "auto route back" for certain messaging groups: Orders, IOIs, Quotes, etc.

See the [FIX Routing Action](#) in the Rules Engine for more details.

## Custom Message Processor

A custom message processor can make routing decisions by using whatever logic the developer chooses — it can be based on message content, data lookup, time of day or any other functionality that is supported in a message processor. Once the target session is determined, routing occurs simply by setting the compID values on the inbound message to match the compID values of the target session.

When routing in this manner, if no matching target session is found, a Reject message is generated and sent back to the session from which the message originated.

## DeliverTo Fields

DeliverTo routing support is built into the FIX specification (see the Session Protocol section of the FIX specification matching your version). It is based on these FIX fields: DeliverToCompID (tag 128), and optionally DeliverToSubID (129) and DeliverToLocationID (145). If the DeliverTo party is defined in the FIX message, then it is used as the routing target. The Node will find a matching outbound session based on those FIX fields.

If no target session match is found, a Reject message is generated and sent back to the session from which the message originated.

When a matching session is found, the following FIX fields are shifted based on this logic:

- Target fields are copied from original DeliverTo fields
- DeliverTo fields are removed
- Sender fields are copied from original Target fields
- OnBehalfOf fields are copied from original Sender fields

### 5.3.4. Routing Side

When Session-Group Routing is used, the Node requires that the `side` attribute be set on all `Session` elements that are configured in a `SessionGroup`. This helps determine which direction a message is heading while routing from one session to another. The `side` attribute can be set to either `buy` or `sell`.

Each `Session` configured in one `SessionGroup` should set its `side` attribute to `buy`, and each `Session` configured in the other `SessionGroup` should set its `side` attribute to `sell`. Not setting these properly can cause FIX routing errors.



## Note

The values `buy` and `sell` are simply labels used to help with the Node's routing logic. They are not related to the messages that are allowed inbound or outbound on a FIX session.

## 5.3.5. Routing Networks

Since FIX 4.3, there are a set of repeating group fields that support multi-hop FIX routing networks. This can be used to track each hub a message has passed through. These fields are: `NoHops` (627), `HopCompID` (628), `HopSendingTime` (629), and `HopRefID` (630).

If the `Link` attribute `useHopFields` is set to `true`, the Node will set these fields accordingly.

## 5.3.6. Handling Routing Issues

### Rejects and RejectMapper

If a routed message gets rejected, there are two ways to handle it: a) It can be logged and dropped, or b) A Reject can be sent back to the originating session. If a Reject is sent to the originator, the `RefSeqNum` field (45) in the Reject message must be set so that it references the correct message at the originator. The Node has a built-in `RejectMapper` for this purpose.

A `RejectMapper` message processor should be configured on the source and listener side of a session as follows:

```
<Sessions>
  <Session counterpartycompid="BROKER" compid="EXCHANGE"
    fixversion="4.4" heartbeat="60" side="sell">
    <Connections>
      <SocketConnection id="sc1" listenport="5001" />
    </Connections>
    <Persister>
      <JournalingPersister id="jpl" baseDirectoryName="./data/persist/broker"/>
    </Persister>
    <SessionManager>
      <SourceMessageProcessors>
        <!-- Initial declaration of RejectMapper on source side -->
        <RejectMapper id="rm_broker" directoryName="./data/rm/rm_broker" />
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <!-- MessageProcessorReference to RejectMapper on listener side -->
        <MessageProcessorReference id="rm_broker_ref" refid="rm_broker" />
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
</Sessions>
```

```

    </SessionManager>
  </Session>
</Sessions>

```

For a complete list of attributes, see the [RejectMapper Configuration Reference](#).

## Buffering

Some planning must be put into the handling of messages when the target session is temporarily unavailable. Most often the two options are to: a) Send a Reject back to the originator, or b) [Buffer messages](#) temporarily until the destination comes back up.



### Caution

If buffering is configured to temporarily hold messages until the destination comes back up, you must consider the types of messages that may get buffered. While it may be okay to buffer post-trade message types it may be very dangerous to buffer time sensitive order messages.

Another type of message buffering/caching feature that is available in the Node is the [Scheduled Order Cache](#). This will intentionally cache order messages for a destination FIX session that is scheduled to be down and will release the messages based on a schedule. This feature has some support for replacing and cancelling orders held in the cache.

## 5.3.7. FIX Routing Configuration

The primary configuration element used in FIX routing is the `Link`, whose structure is shown here:

```

<Link id="routing_table">
  <SessionGroup>
    <SessionReference id="ref1" counterpartycompid="CLIENT1" compid="ROUTER"/>
    ...
  </SessionGroup>
  <SessionGroup>
    <SessionReference id="ref2" counterpartycompid="EXCH1" compid="ROUTER"/>
    ...
  </SessionGroup>
</Link>

```

Only one `Link` is defined per instance. It optionally contains two `SessionGroup` elements, which contain one or more `SessionReference` elements that reference each applicable `Session`.

For a complete list of attributes for these elements, see the [Link Configuration Reference](#) and [SessionReference Configuration Reference](#).



## Note

If either the Rules Engine or a custom message processor is used for routing, the `useDeliverToRouting` attribute of the `Link` must be set to `false`. Otherwise there can be contention between the `DeliverTo` routing fields and the Rules Engine or message processor routing logic.

As explained [earlier](#), the router can be configured to operate in one of two routing modes:

## Session-Group Routing Configuration

The `Link` element must contain two `SessionGroup` elements which define the two sets of sessions participating in routing. The `side` attribute on each `Session` must also be correctly set.

```
<Application id="router">
  <Link id="routing_table">
    <SessionGroup>
      <SessionReference id="sessionref1" counterpartycompid="CLIENT1" compid="BROKER_ABC"/>
      <SessionReference id="sessionref2" counterpartycompid="CLIENT2" compid="BROKER_ABC"/>
    </SessionGroup>
    <SessionGroup>
      <SessionReference id="sessionref3" counterpartycompid="EXCH1" compid="BROKER_ABC"/>
      <SessionReference id="sessionref4" counterpartycompid="EXCH2" compid="BROKER_ABC"/>
    </SessionGroup>
  </Link>
  <Sessions>
    <!-- Remember to set each Session's 'side' attribute correctly -->
    <Session counterpartycompid="CLIENT1" compid="BROKER_ABC" side="buy" fixversion="4.2">
      ...
    </Session>
    <Session counterpartycompid="CLIENT2" compid="BROKER_ABC" side="buy" fixversion="4.4">
      ...
    </Session>
    <Session counterpartycompid="EXCH1" compid="BROKER_ABC" side="sell" fixversion="4.2">
      ...
    </Session>
    <Session counterpartycompid="EXCH2" compid="BROKER_ABC" side="sell" fixversion="4.4">
      ...
    </Session>
  </Sessions>
</Application>
```

## Routing Hub Configuration

The `anyToAny` and `ignoreSessionSide` attributes of the `Link` must both be set to `true`. No `SessionGroup` elements are required, nor is it necessary to set the `side` attribute for each `Session`.

```

<Application id="router">
  <!-- Link element defined at Application level, the 'anyToAny' and
       'ignoreSessionSide' attributes must be set to 'true' in this
       setup and SessionGroup elements aren't required -->
  <Link id="routing_hub" anyToAny="true" ignoreSessionSide="true"/>
  <Sessions>
    <!-- Session's 'side' attribute not required to be set in this config -->
    <Session counterpartycompid="CLIENT1" compid="BROKER_ABC" fixversion="4.2">
      ...
    </Session>
    <Session counterpartycompid="CLIENT2" compid="BROKER_ABC" fixversion="4.4">
      ...
    </Session>
    <Session counterpartycompid="EXCH1" compid="BROKER_ABC" fixversion="4.2">
      ...
    </Session>
    <Session counterpartycompid="EXCH2" compid="BROKER_ABC" fixversion="4.4">
      ...
    </Session>
  </Sessions>
</Application>

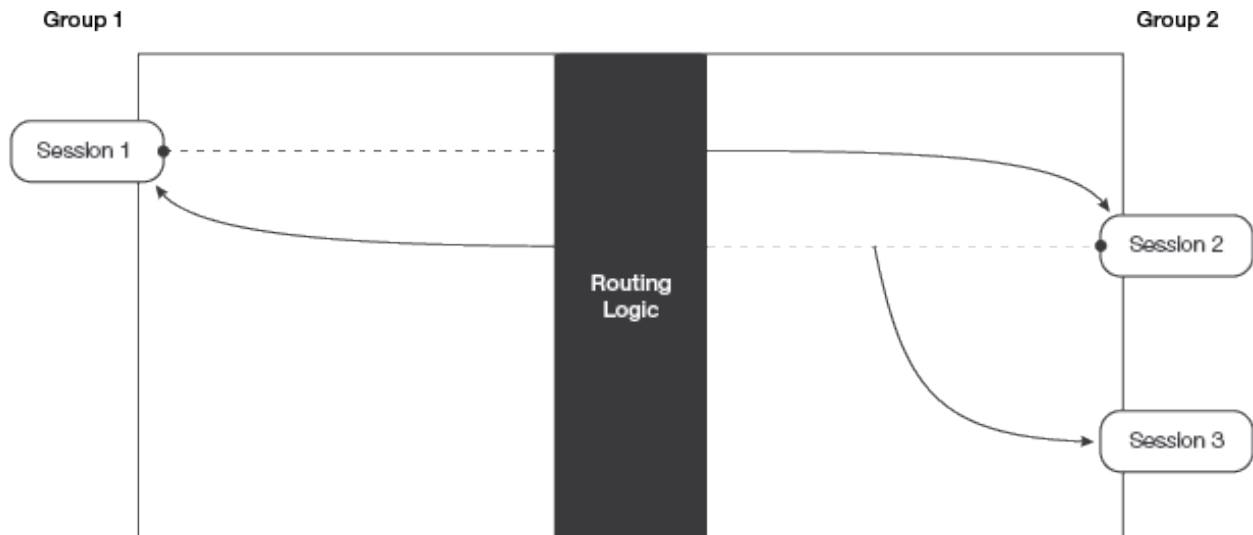
```

## 5.4. FIX Drop Copy

### 5.4.1. Introduction

The Node supports the "drop copy" of FIX messages. In general, to drop copy a message is to copy a FIX message from one FIX session and send it outbound on another FIX session. The copied message can be an exact duplicate, or it can be modified by adding, removing or altering tags.

A common use case for drop copy is for a broker to copy all fills on live trading FIX sessions and send them to an internal risk system.





## 5.4.2. Drop Copy Methods

### Rules Engine

The [Catalys Rules Engine](#) has built-in drop copy support. Drop copy rules of varying complexity can be created to meet your requirements by taking advantage of the built-in conditions and expression language. Multiple copies of one message can be made and sent outbound on any number of sessions.

See the [Drop Copy Action](#) in the Rules Engine for more details.

### Custom Message Processor

A custom message processor can be used to make a clone of the original message, modify it and send it out one or more other sessions.

If no session match is found, a Reject is generated and sent back to the inbound session where the original message came in on.

## 5.5. Message Buffering

### 5.5.1. Introduction

Buffering provides a mechanism to store messages which are temporarily undeliverable. When buffering is not configured on a session, the Node will reject messages which cannot be delivered. Some common reasons why a message cannot be delivered are:

- A message arrives from an internal application over a technology adapter but the outbound session is not logged on.
- A message arrives from a session but the internal application is unreachable.
- A message arrives from a session in a FIX routing configuration but the outbound session is down.
- A message arrives from a session, a [drop copy](#) is made and the drop copy's target session is down.

The default behaviour of rejecting undeliverable messages may not be desirable. Instead, you may prefer to save undeliverable messages on disk until the target session becomes available.

Buffered messages are stored on disk in a [persistent collection](#). This means that buffered messages are retained after restarts or unexpected termination of the Node, and they are replicated across members of a high-availability cluster.

### 5.5.2. Configuration

Message buffers are built-in message processors that can be configured and nested like any other message processor.

## Source Message Buffer

The `SourceMessageBuffer` is used to buffer undeliverable outbound messages, usually because the session is down.

Generally, the `SourceMessageBuffer` is the outermost message processor in the `SourceMessageProcessors` chain:

```
<SessionManager>
  <SourceMessageProcessors>
    <!-- SourceMessageBuffer configured as the outermost message processor -->
    <SourceMessageBuffer id="smb1" directory="./data/buffer">
      <CatalysRulesEngine id="cre1" rulesPackId="outbound-enrichment-rules">
        <JmsConsumer id="jms1" type="queue" destination="BUY_OUT" factory="primaryQCF"/>
      </CatalysRulesEngine>
    </SourceMessageBuffer>
  </SourceMessageProcessors>
  <ListenerMessageProcessors/>
</SessionManager>
```

For a complete list of attributes, see the [SourceMessageBuffer Configuration Reference](#).

For a working demonstration of message buffering, see the Buffering template located in *templates/Buffering/Simple*.

## Listener Message Buffer

The `ListenerMessageBuffer` is used to buffer undeliverable inbound messages, usually due to outages in any downstream internal applications, e.g. a message queue.

Generally, the `ListenerMessageBuffer` is configured directly above a technology adapter in the `ListenerMessageProcessors` chain:

```
<SessionManager>
  <ListenerMessageProcessors>
    <CatalysRulesEngine id="cre1" rulesPackId="inbound-rules">
      <!-- ListenerMessageBuffer configured directly above technology adapter -->
      <ListenerMessageBuffer id="lmb1" directory="./data/buffer">
        <JmsConsumer id="jms1" type="queue" destination="BUY_IN" factory="primaryQCF"/>
      </ListenerMessageBuffer>
    </CatalysRulesEngine>
  </ListenerMessageProcessors>
  <SourceMessageProcessors/>
</SessionManager>
```

For a complete list of attributes, see the [ListenerMessageBuffer Configuration Reference](#).

### 5.5.3. Persistent Collection Configuration

The underlying persistent collection of each buffer can be tuned as necessary. Refer to [Persistent Collection Properties](#) for a list of attributes, which are configured as follows:

```
<SessionManager>
  <ListenerMessageProcessors>
    <!-- ListenerMessageBuffer configured directly above technology adapter -->
    <ListenerMessageBuffer id="lmb1" directory="./data/buffer/">
      <!-- Properties for lmb1 -->
      <Properties id="lmb1_props">
        <Property name="rollJournalFileEvery" value="4294967296" />
      </Properties>
      <JmsConsumer id="jms1" type="queue" destination="BUY_OUTBOUND" factory="primaryQCF"/>
    </ListenerMessageBuffer>
  </ListenerMessageProcessors>
  <SourceMessageProcessors/>
</SessionManager>
```

### 5.5.4. Managing Buffers from the Command Line

Message buffers have a set of standard commands that can be executed from the Node's [command-line interface](#) (CLI).

Each command accepts the name of a buffer as an optional argument. If this argument is omitted, the command operates on all configured buffers. The names of all buffers can be displayed using the `c_list` command.

Command	Arguments	Description
<code>mb_clear</code> ( <code>mb_c</code> )	<code>&lt;buffer_name   all&gt;</code>	Clear out all messages in the specified buffer(s).
<code>mb_error</code> ( <code>mb_e</code> )	<code>&lt;buffer_name   all&gt;</code>	Display the errors (if any) which caused buffering to commence.
<code>mb_messages</code> ( <code>mb_m</code> )	<code>&lt;buffer_name [range]   all&gt;</code>	Display the messages in the specified buffer(s).  By default, the oldest 100 messages are displayed. Use the optional <code>range</code> argument to show either the oldest <code>n</code> messages, or a range of messages, <code>n-m</code> .
<code>mb_remove</code> ( <code>mb_r</code> )	<code>&lt;buffer_name [range]   all&gt;</code>	Removes elements from the front of the buffer or in a range from first to last ( <code>n-m</code> ).

Command	Arguments	Description
mb_size (mb_s)	<buffer_name   all>	Display the size of the specified buffer(s).

## 5.6. Message Throttling

### 5.6.1. Introduction

Throttling provides a mechanism to allow only a certain number of messages per a configurable time period, normally a second, pass through a FIX session. When throttling is not configured, the Node processes FIX messages as fast as possible.

Some common situations when throttling is utilized:

- An internal application is a slow consumer of inbound messages, so [listener](#) throttling can be configured to limit the inbound message rate.
- A FIX counterparty can only consume a certain rate of messages, so [source](#) throttling can be configured to limit the outbound message rate.
- An exchange allows only a certain number of orders per second to be sent from each client, so [listener](#) throttling can be configured on all client sessions to operate at the same inbound rate.

One way the Node throttles messages is by storing them temporarily on disk before delivering further upstream or downstream. This means that throttled messages are retained after restarts and unexpected termination of the Node, and they are replicated across members of a high-availability cluster.

Alternatively, the Node can throttle inbound messages by governing the read operations on the FIX session's inbound socket when the sender exceeds the permitted rate limit. This is a more efficient and accurate way to throttle messages — the application does not waste resources (e.g. CPU and disk I/O) on messages that are sent too quickly. See [Inbound Socket Throttling](#) for a configuration example.

### 5.6.2. Configuration

Message throttling is a built-in feature of the Node in the form of a message processor that can be configured and nested like any other message processor.

#### Source Message Throttling

The `SourceMessageThrottle` is used to throttle **outbound** messages on a session at a certain rate. It is usually configured as the outermost message processor on the Source message processing chain:

```
<SessionManager>
  <SourceMessageProcessors>
    <!-- SourceMessageThrottle configured as the outermost message processor. This
```

## Standard Features

```
will throttle outbound message flow to 100 messages per second -->
<SourceMessageThrottle id="throttle" directory="./data/throttle"
    messagesPerInterval="100" interval="1">
  <CatalysRulesEngine id="cre" rulesPackId="outbound-enrichment-rules">
    <JmsConsumer id="jms" type="queue" destination="BUY_OUTBOUND" factory="primaryQCF"/>
  </CatalysRulesEngine>
</SourceMessageThrottle>
</SourceMessageProcessors>
<ListenerMessageProcessors/>
</SessionManager>
```

In cases where a strict outbound message rate limit is required even when messages are resent, [resendThrottleInterval](#) and [resendThrottleMessagesPerInterval](#) attributes of the `Session` element should be used.

For a complete list of attributes, see the [SourceMessageThrottle Configuration Reference](#).

## Listener Message Throttling

The `ListenerMessageThrottle` is used to throttle **inbound** messages on a session at a certain rate. It is usually configured as the outermost message processor on the Listener message processing chain:

```
<SessionManager>
  <ListenerMessageProcessors>
    <!-- ListenerMessageThrottle normally the outer most configured message processor. This
    will throttle inbound message flow at 100 messages per second -->
    <ListenerMessageThrottle id="throttle" directory="./data/throttle"
      messagesPerInterval="100" interval="1">
      <CatalysRulesEngine id="cre" rulesPackId="inbound-rules">
        <JmsConsumer id="jms" type="queue" destination="BUY_OUTBOUND" factory="primaryQCF"/>
      </CatalysRulesEngine>
    </ListenerMessageThrottle>
  </ListenerMessageProcessors>
  <SourceMessageProcessors/>
</SessionManager>
```

For a complete list of attributes, see the [ListenerMessageThrottle Configuration Reference](#).

For a working demonstration of inbound message throttling to disk, see the Throttling template located in *templates/Throttling/Persisted*.

## Inbound Socket Throttling

To enable inbound throttling at the FIX session's socket layer, set the [maximumInboundMessageRate](#) attribute on the `Session` to the maximum number of messages to read from the socket per second. Throttling is applied prior to reading messages from the TCP socket. Messages are not buffered or queued in the application; rather, when a sender exceeds the configured rate, messages will accumulate in the inbound socket's buffer. Note this is not supported when `multiplexingMode` is configured.

```
<!-- throttles inbound messages on this session to 500 messages/sec -->
```

```
<Session maximumInboundMessageRate="500"
  counterpartycompid="CLIENT" compid="SERVER" fixversion="4.4" >
  <Connections>
    <SocketConnection id="sc" listenport="2002"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors/>
    <ListenerMessageProcessors/>
  </SessionManager>
</Session>
```

For a working demonstration of inbound message throttling at the socket layer, see the Throttling template located in *templates/Throttling/InboundSocket*.

## 5.7. FIX Dictionaries

### 5.7.1. Introduction

FIX dictionaries provide a way to define and access FIX field and message information. This is particularly useful in message transformers that need to access information about field tags, names, data types and expected values. A set of default dictionaries is bundled with the Node, but they can be customized as needed.

Features of the dictionaries include:

- Simple XML-based implementation
- Customizable for use in transformers and other custom code
- Open interface for custom implementations
- Access to message structure information
- Access to message field and field group information
- Access to field value information

### 5.7.2. Getting Started

The FIX dictionaries define field groups for each message type for each FIX version specification. These dictionaries are automatically generated from the FPL FIX repositories.

The dictionary files are packaged inside the *catalys-node.jar* library. If you do not need to add custom dictionaries or modify the dictionaries there is nothing you have to do — the Node will find and use the proper dictionary inside the jar by default.

A copy of the dictionary files can be found in the *resources/dictionary* directory of the Node distribution. The dictionaries are in an XML format specified by *dictionary.dtd*, which can be found in the same directory.

You can use this copy as a starting point if you need to make modifications or create your own custom dictionaries.

### 5.7.3. Custom Dictionaries

The preferred method of writing a custom dictionary is to use XML.

To define your custom fields, field groups, or messages in a custom XML dictionary the following steps are required:

1. Create a new XML file (e.g. *fix.4.4.cust.xml*) in the *resources/dictionary* directory.
2. Add the new definitions to this file according to the format given in *resources/dictionary.dtd*.
3. Prepend the *resources* directory to the Java classpath of the Node application so that it can find your custom file at runtime.

To enable extra debugging when loading and using your custom dictionary, set the [loggers](#) for *com.camerontec.catalys.util.dictionary* to `DEBUG` level. All information regarding the creation of fields, field groups and messages will then be sent to those loggers.

### 5.7.4. Configuration of Custom Dictionary

Once the new dictionary has been created and included in the classpath of the application, it can be configured into the Node by setting the `dictionaryName` attribute of the `Session` element:

```
<Session dictionaryName="fix.4.4.cust" ...>
  ...
</Session>
```

### 5.7.5. Dictionaries API

Custom code can access the new dictionary via the [XMLDictionaryFactory.getDictionary\(name\)](#) method. For example:

```
IDictionary myDictionary = XMLDictionaryFactory.getInstance().getDictionary("fix.4.4.cust");
```

The [IDictionary](#) interface provides methods to get the field, field group and message information associated with the dictionary.

### 5.7.6. Non XML-based Dictionaries

To make non XML-based dictionaries available to your custom components (e.g. message processor) you need to write an implementation of [IDictionaryFactory](#) which reads/produces the dictionaries from an alternate source.

You can then use this factory and the dictionaries it produces in your own code. Note that it is not possible to override the XML dictionaries used by the Node for internal purposes.

## 5.8. Inbound Message Validation

### 5.8.1. Introduction

The Catalys Node enforces several levels of validation on inbound FIX messages. This document describes the different checks that are performed on each inbound messages, how the Node responds when each check fails and how to disable the check (if allowed).

### 5.8.2. Validation Rules

Problem Description	FIX Validation Type	Node Response	Setting to Disable
The BeginString field isn't at the first position in a message.	Garbled Message	Message will be discarded silently. After 2 consecutive garbled messages, session will be logged out.	-none-
BodyLength field isn't at the second position in a message.	Garbled Message	Message will be discarded silently. After 2 consecutive garbled messages, session will be logged out.	-none-
MsgType field isn't at third position in a message.	Garbled Message	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	<p>This check, and other checks, can be disabled by setting the <code>trustedMode</code> attribute to <code>true</code> on a <a href="#">Session</a> element.</p> <p>This check alone can be disabled by setting the <code>dialectIgnore-TagOrdering</code> attribute to <code>true</code>.</p>
Message BodyLength is incorrect.	Garbled Message	Message will be discarded silently. After 2 consecutive garbled messages, session will be logged out.	-none-



## Standard Features

Problem Description	FIX Validation Type	Node Response	Setting to Disable
Message has incorrect ending sequence (i.e. 10=###<SOH>).	Garbled Message	Message will be discarded silently. After 2 consecutive garbled messages, session will be logged out.	-none-
Message CheckSum is incorrect, FIX bytes representing a CheckSum cannot be converted to a positive integer or the calculated CheckSum does not match the one provided in the message.	Garbled Message	Message will be discarded silently. After 2 consecutive garbled messages, session will be logged out.	-none-
Message Signature wrong type of data or couldn't be converted to bytes.	Message Integrity Issue	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	-none-
Exception during Signature decryption.	Message Integrity Issue	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	-none-
Exception during message field decryption.	Message Integrity Issue	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	-none-

## Standard Features

Problem Description	FIX Validation Type	Node Response	Setting to Disable
Message has a field without a value.	Message Parsing Issue	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	-none-
Message bytes representing a field cannot be converted to a positive integer.	Message Parsing Issue	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	-none-
Message bytes have an encoded field but no corresponding MessageEncoding field.	Message Parsing Issue	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	-none-
Message has <code>DATA</code> field data type without preceding data length field.	Message Parsing Issue	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	This check can be disabled by for the RawData field (tag96) by setting the <code>dialectAllowNoLengthOnRawData</code> attribute to <code>true</code> .
Message has <code>DATA</code> field preceded by invalid data length field or cannot be converted to a positive integer or is missing the field delimiter character (SOH).	Message Parsing Issue	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	This check can be disabled by for the RawData field (tag96) by setting the <code>dialectAllowNoLengthOnRawData</code> attribute to <code>true</code> .

## Standard Features

Problem Description	FIX Validation Type	Node Response	Setting to Disable
Message missing field delimiter character (SOH) at the end of a message bytes stream.	Message Parsing Issue	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	-none-
Message missing one or more of the following fields: 8, 9, 35, 52, 49, 56, 34, 10.	Message Validation Issue	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	This check, and other checks, can be disabled by setting the <code>trustedMode</code> attribute to <code>true</code> on a <a href="#">Session</a> element.
Message field is missing a value.	Message Validation Issue	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	<p>This check, and other checks, can be disabled by setting the <code>trustedMode</code> attribute to <code>true</code> on a <a href="#">Session</a> element.</p> <p>This check alone can be disabled by setting the <code>dialectIgnore-TagOrdering</code> attribute to <code>true</code>.</p>
Message BeginString value in Logon message does not match the FIX version configured on the <code>Session</code> element.	Message Validation Issue	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	<p>This check, and other checks, can be disabled by setting the <code>trustedMode</code> attribute to <code>true</code> on a <a href="#">Session</a> element.</p> <p>This check alone can be disabled by setting the <code>dialectIgnore-BEGINSTRING</code> attribute to <code>true</code>.</p>

## Standard Features

Problem Description	FIX Validation Type	Node Response	Setting to Disable
Message received that BeginString value that is not consistent with other messages on a single session.	Message Validation Issue	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	This check can be disabled by setting the <code>dialectIgnore-BEGINString</code> attribute to <code>true</code> .
Timestamp message fields 52 and 122 are a format that is not a supported FIX timestamp format.	Message Validation Issue	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	<p>This check, and other checks, can be disabled by setting the <code>trustedMode</code> attribute to <code>true</code> on a <a href="#">Session</a> element.</p> <p>This check alone can be disabled by setting the <code>IgnoreTimestampFormat</code> attribute to <code>true</code>.</p>
Message received with a sender or target compID that is not consistent with other messages on a single session.	Message Validation Issue	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	<p>This check, and other checks, can be disabled by setting the <code>trustedMode</code> attribute to <code>true</code> on a <a href="#">Session</a> element.</p> <p>This check alone can be disabled by setting the <code>dialectIgnoreCompanyID</code> attribute to <code>true</code>.</p>
Message fields 43 or 97 does not contain Y or N value.	Message Validation Issue	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	This check, and other checks, can be disabled by setting the <code>trustedMode</code> attribute to <code>true</code> on a <a href="#">Session</a> element.

Problem Description	FIX Validation Type	Node Response	Setting to Disable
Message exceeds provided maximum message length in bytes.	Message Validation Issue	Session is disconnected.	To configure the maximum message length, use the <code>maxLength</code> attribute on the <a href="#">Session</a> element. By default this is 1MB.
Message <code>MsgType</code> value is invalid for the FIX version of the Session. If the <code>MsgType</code> value begins with 'U' then it's ignored.	Message Validation Issue	Message will be discarded and Logout will be sent in case when incoming Logon message is garbled. In all other cases a Session Reject will be sent to the counterparty.	This check is disabled by default. To enforce, set the <code>validateMessageTypes</code> attribute to <code>true</code> and the <code>trustedMode</code> attribute to <code>false</code> on a <a href="#">Session</a> element.

## 5.9. Message Factories

### 5.9.1. Introduction

Message Factories are used by sessions to create `IFIXMessage` message objects. Two typical scenarios for using non-default message factories are: adding non-standard fields to a specific session layer message type, and controlling the type of `IFIXMessage` object that is created. For example, the `FastFriendlyFIXMessageFactory` creates a special implementation of `IFIXMessage` optimized for sending FIX messages over a FAST-encoded session.

The default message factories are [DefaultFromFIXMessageFactory](#) and [DefaultToFIXMessageFactory](#) which create messages of type [FIXMessageAsIndexedByteArray](#) and [FIXMessageAsArray](#) respectively. However, it is possible to configure a session to use a different message factory. There are a number of [provided message factory implementations](#) and it is possible to develop a [custom implementation](#).

### 5.9.2. Configuration

The `MessageFactories` configuration element is used to define message factory configurations. Each configured factory must have an `id` attribute, which is referenced by the `toFIXMessageFactoryRefId` and `fromFIXMessageFactoryRefId` attributes of `Session` elements to indicate which message factory should be used. For example:

```
<Application id="CatalysSell">
```

```

<MessageFactories>
  <ConfigurableMessageClassMessageFactory
    id="inboundFactory"
    messageClass="com.camerontec.catalys.core.message.FIXMessageAsIndexedList"
  />
  <ConfigurableMessageClassMessageFactory
    id="outboundFactory"
    messageClass="com.camerontec.catalys.core.message.FIXMessageAsIndexedByteArray"
  />
</MessageFactories>
<Sessions>
  <Session counterpartycompid="BROKER"
    compid="MARKET"
    fixversion="4.4"
    fromFIXMessageFactoryRefId="inboundFactory"
    toFIXMessageFactoryRefId="outboundFactory">
    ...
  </Session>
</Sessions>
</Application>

```

### 5.9.3. Provided Implementations

Besides the default message factories, the Node includes a number of alternate factories:

#### ConfigurableMessageClassMessageFactory

This message factory returns instances of any class which implements `IFIXMessage`. The message class to use is specified via its `messageClass` attribute.

Additionally, this message factory adds the Username(553) and Password(553) fields to Logon messages if the optional `username` and `password` attributes are set. While some implementations need dedicated factories with special configuration elements, simpler implementations like `FIXMessageAsIndexedByteArray` and `FIXMessageAsArray` do not need a dedicated factory and can be set by using the `ConfigurableMessageClassMessageFactory` element.

For a complete list of attributes, see the [ConfigurableMessageClassMessageFactory Configuration Reference](#).

#### CMEiLink2MessageFactory

This message factory is required to connect to the iLink 2.0 FIX gateway at the CME. The gateway requires the RawData(96) field be present in Logon messages. Messages created using this factory will contain that field populated automatically as well as the associated RawDataLength(95) field.

For a complete list of attributes, see the [CMEiLink2MessageFactory Configuration Reference](#).

## FastFriendlyFIXMessageFactory

This message factory creates `IFIXMessage` objects that are optimized for use over a FAST-encoded session. Messages created by this factory have an order imposed by the FAST template associated with each message type. Hence, addition of fields to these messages are guaranteed to be inserted in the correct order.

For a complete list of attributes, see the [FastFriendlyFIXMessageFactory Configuration Reference](#).

### 5.9.4. Custom Implementation

As stated above, it is possible to develop your own custom message factory. Please review the [Message Factory Programming Guide](#) for instructions. To configure your custom message factory, see the following example which uses the `GenericMessageFactory`.

```
<Application id="CatalysSell">
  <MessageFactories>
    <GenericMessageFactory
      id="outboundFactory"
      class="com.camerontec.catalys.core.message.CustomMessageFactory">
      <Properties id="outboundFactoryProps">
        <Property name="prop1" value="value1"/>
      </Properties>
    </GenericMessageFactory>
  </MessageFactories>
  <Sessions>
    <Session counterpartycompid="BROKER"
      compid="MARKET"
      fixversion="4.4"
      toFIXMessageFactoryRefId="outboundFactory">
      ...
    </Session>
  </Sessions>
</Application>
```

For a complete list of attributes, see the [GenericMessageFactory Configuration Reference](#).

# Chapter 6. Optional Features

## Table of Contents

6.1. High Availability .....	126
6.1.1. Introduction .....	126
6.1.2. Role Determination .....	127
6.1.3. Failover .....	127
6.1.4. State Persistence .....	129
6.1.5. Single IP Address .....	131
6.1.6. Configuration .....	134
6.1.7. High Availability Operations .....	139
6.1.8. High Availability Communications .....	144
6.2. Catalys Rules Engine .....	148
6.2.1. Introduction .....	148
6.2.2. Configuration .....	149
6.2.3. Expression Language .....	152
6.2.4. Conditions .....	157
6.2.5. Actions .....	162
6.2.6. Message Validation .....	177
6.3. File Lookup Service .....	178
6.3.1. Introduction .....	178
6.3.2. Basic Configuration Example .....	178
6.3.3. Configuring Multiple Keys and Results .....	179
6.3.4. Lookup from a Custom Message Processor .....	180
6.3.5. Labels .....	181
6.4. Scheduled Order Cache .....	182
6.4.1. Introduction .....	182
6.4.2. Configuration .....	182
6.4.3. Behavior of Scheduled Order Cache .....	184
6.5. FIXML Transformer .....	184
6.5.1. Introduction .....	184
6.5.2. Configuration .....	185
6.5.3. Customizing FIXML Definitions - FixmlElementGenerator .....	186
6.5.4. Templates .....	187
6.6. SSL Socket Connection .....	187
6.6.1. Introduction .....	187
6.6.2. Java Keystores and Truststores .....	188
6.6.3. SSL Protocols .....	188
6.6.4. Configuration .....	189
6.6.5. Client Authentication .....	192



6.6.6. Troubleshooting .....	194
6.7. Data Encryption .....	195
6.7.1. Introduction .....	195
6.7.2. Configuration .....	195
6.7.3. Encryption Keys .....	197
6.7.4. Troubleshooting .....	197
6.8. Breakout Box .....	198
6.8.1. Introduction .....	198
6.8.2. Concepts .....	198
6.8.3. Examples .....	199
6.8.4. Configuration .....	204
6.9. Market Compliance .....	204
6.9.1. Introduction .....	204
6.9.2. Compliance Filters .....	205
6.9.3. Operational Model .....	207
6.9.4. Configuration .....	207
6.9.5. Rules File .....	208
6.9.6. Price Bands for Percentage Filters .....	220
6.9.7. Price Step Table File .....	222
6.9.8. Extending Filters .....	222
6.9.9. The <code>absolute</code> Attribute .....	223
6.10. Message Compression .....	223
6.10.1. FAST Support .....	223
6.10.2. FAST and the FIX Libraries .....	224
6.10.3. Additional Notes .....	224
6.11. Market Data Services .....	224
6.11.1. Introduction .....	224
6.11.2. Configuring a Market Data Service .....	225
6.11.3. Using a Market Data Service .....	226
6.11.4. Writing a Market Data Service .....	226
6.11.5. Writing a Derived Market Data Service .....	227

## 6.1. High Availability

### 6.1.1. Introduction

Catalys High Availability (HA) is an optional feature that provides operational continuity when an outage occurs in a Catalys Node or its operating environment. In an HA deployment (a *cluster*), two or more Catalys Nodes are configured to operate as a single, logical, Catalys Node.

Once a steady state is reached in an  $N$  Node cluster, at any point in time there is:

- One Node assigned the role of *primary*. Only the primary Node maintains live FIX sessions.

- $N-1$  Nodes assigned the role of *secondary*. Secondary Nodes are kept synchronized with the primary Node so that they are available to assume the role of primary should the current primary fail.
- One cluster IP address to which FIX counterparties may connect. This is generally achieved via an IP takeover mechanism (the Node provides lifecycle hooks to facilitate this) or managed by a network device (e.g. load balancer or content switch) capable of detecting which Node is currently primary and directing IP traffic accordingly.

Before continuing on to the more high level aspects of HA as described in the next sections, you may consider skipping to the [HA Communications](#) section to gain an understanding of how HA Nodes communicate with each other, why, and the various options available. An understanding of these topics is likely to help in choosing the right parameters for, and setting expectations for the behavior of, your cluster.

### 6.1.2. Role Determination

Each Node assigns itself a role based on the following criteria:

- Primary priority: A Node's primary priority is inferred from its ordinal position in the list of all configured [ClusterNode](#) elements in the [HighAvailabilityCluster](#) element. Those which appear earlier in the configuration have a higher primary priority than those appearing later.
- Presence and Broadcasted Roles of Other Nodes: Every Node periodically broadcasts its current role (and therefore declares its presence) to all other Nodes. Each Node is therefore able to build a picture of the state of the entire cluster before determining which role it should assign to itself.

Nodes continue to periodically broadcast their roles to all other Nodes after initial role assignment so that all other Nodes can maintain a consistent picture of the state of the cluster. This serves two other purposes:

1. Detection of a Node Leaving the Cluster: If a Node's role is not refreshed in a timely manner, the `groupAgeThreshold`, it is regarded as having left the cluster.
2. Detection of "Split Brain": This situation can occur when more than one Node declares itself as the Primary Node. It indicates an erroneous operational state and causes all Nodes in the cluster to participate in the selection of a new primary Node.

### 6.1.3. Failover

#### Automatic Failover

Automatic failover is when a secondary Node in the cluster assigns itself the primary role either because the primary Node is no longer declaring its presence (or the broadcasts are not being received). By default this occurs automatically when the current primary can no longer run the tasks required of it, for example due to a hardware or software failure, a network outage, a slow-running Node or other causes. In the case of network failures the decision to failover is far from trivial. Failover needs to occur in a timely

fashion as well as for the right reasons (a true failure, rather than a Node itself becoming isolated from the network).

When a Node configured for automatic failover (the default) detects there is no longer any primary Node in the cluster, it first determines whether it should attempt to become the new primary based on its primary priority (see above). If a Node determines that it is the highest priority primary candidate then it will first run a number of [primary validity checks](#) before assuming the primary role.

If all primary validity checks pass, then the [becoming primary actions](#) are executed. The most common actions are:

- Perform a takeover of the cluster's IP address by the local Node
- Force upstream devices to update their ARP tables to direct IP packets to the new primary Node.



### Note

The above actions describe the scenario when IP takeover is used to reassign the cluster's IP address to the new primary Node. If another mechanism is being used (e.g. load balancer or content switch) then no becoming primary actions are required.

Once the Node has completed becoming the primary, it begins broadcasting its new role to the rest of the cluster. If the previous primary returns to the cluster at some point later, it will not attempt to regain the primary role despite having a higher primary priority.

## Manual Failover

Manual failover can be initiated at any cluster Node. Typically, Nodes at remote sites are configured not to failover automatically (since it usually indicates a catastrophic failure at the primary site) and therefore may only assume the primary role via a manual failover.

Manual failover can also be used to restore a previously failed primary Node to be the current primary once again.

A manual failover can be performed in the following ways:

- Issuing the [cl primary](#) command via the [remote command line](#), command line exposed on the Dashboard or on a Node instance running in console mode.
- Clicking the "Make Primary" button on the instance you want to failover to in the Server panel of the Dashboard.
- Programmatically from the JMX API.

## 6.1.4. State Persistence

To be highly available, the state of the primary Node must be persisted in such a way that no single failure will cause the system to stop processing. The Node uses a replication mechanism that distributes every persistent data update from the primary to each secondary. Depending on the [write quorum](#) configuration, each update will be acknowledged by zero or more secondary Nodes before the next data update is processed.

### Synchronization

When a Node joins the HA cluster, it automatically synchronizes persistent data with the other members. The first phase involves querying the state of the other Nodes to determine if it has the latest data. If it is missing any updates, it transitions to a synchronizing state until it has an exact copy of latest data. While a Node is synchronizing it cannot become the primary.

Note that state synchronization is bi-directional by default. That is, Nodes already in the cluster also query the joining Node for state updates. For this reason it is critical to ensure that any data which exists on the joining Node is valid because it is possible that it be propagated back to other Nodes in the cluster as missing updates. The most likely reason for this to happen is an operational error whereby the previous day's data is reset at the primary site but not at the DR site. At startup the next day, the DR sees itself as being "ahead" of the other Nodes and (potentially stale) data is propagated back to the other Nodes.

There are internal checks in the software and also some configuration attributes which help protect against this and other similar unexpected outcomes:

- [processReplayRequests](#)
- [synchronizationWindow](#)

As implied above, data synchronization between Nodes is *symmetric* by default. That is, by default, all Nodes treat all other Nodes as potential sources of missing data and actively try to retrieve any missing data. This is not always the expected behavior, especially for DR Nodes. The `processReplayRequests` attribute is a way of changing this behavior on a per Node basis. When explicitly set to `false` (it is `true` by default), that Node never reports or propagates missing data to any other Nodes in the cluster. In some systems this fits most closely with the notion of a DR Node and for this reason *it is strongly recommended* that this option be enabled for DR Nodes.

Though not strictly related to data integrity, the `synchronizationWindow` attribute exists to protect the cluster from inadvertently introducing high synchronization overhead into a running cluster during high throughput. This can happen if, for example, a Node with no persistent data is introduced into the cluster late in the trading day. In this case the joining Node is by definition "behind" other Nodes and requires synchronization of all data. Depending on their size and the current system throughput this can place enough additional overhead on the system to disrupt FIX message traffic. To protect against this it is possible to specify a non-zero `synchronizationWindow` which represents the maximum number of "missed" updates (per collection) that will be serviced by another Node. That is, if the local Node is too

far "behind" (defined by the `synchronizationWindow`) then it never completes initial synchronization and never joins the cluster.

The above configuration attributes are useful to help prevent cluster instability or degraded performance due to what would normally be considered operational errors. There are times though when they actually hinder system administration. For example:

- Installing a new Node into a cluster, during off hours, with live data. In this case the `synchronizationWindow` can actually prevent the cluster's synchronization mechanism from doing all the hard work.
- Rebuilding data on a (preferred) primary, during off hours, following a manual failover to the DR during the previous day's off hours. In this case, the DR is likely to have the `processReplayRequests` attribute set to `false` and therefore will not service the (preferred) primary Node's request to replay data.

For these reasons it is possible to manually override the configured behavior of the above options, when it is safe to do so, in order to temporarily re-enable bi-directional and large data synchronization. This is achieved via the command line. See the `cl_replay_on`, `cl_replay_off`, `cl_sync_win` and `cl_restore_defs` commands, from the set of available [cluster console commands](#).

## Cluster Site

Each Node in the HA cluster belongs to a logical *site*, which in most cases is defined by a physical location. For enterprise environments, it's common to have a local site and a disaster recovery site. The Nodes in a site share the definition for the number (and source) of acknowledgements that are required before a transaction is said to be replicated. This definition is called a *write quorum* specification.

From the perspective of each Node, its local site is the site to which it belongs, and all others are considered remote sites.

## Write Quorum

The write quorum defines the number of explicit acknowledgements that must be received by the primary for each write request before it processes the next write request for the same data set. Requiring a non-zero write quorum therefore implies an additional latency at least equal to the round trip time for the write request and its acknowledgement. Receipt of an acknowledgement to a particular write request implies only that the request has been received by another Node, not necessarily that it has been processed. In general the higher the effective write quorum the higher the certainty that no write requests are lost (i.e. more concurrent system failures can be tolerated). Refer to the [Write Quorum Settings](#) section which explains how the effective write quorum is calculated based on the configured settings.

On the other hand, not using the write quorum means that write request replication is on a "best effort" basis only and it is likely (especially in high throughput systems) that secondary Nodes run "behind" the primary Node. In the context of a FIX session, should a failover occur to a secondary Node which is "behind" the old primary the following types of behavior can be observed:

- When counterparties reestablish the session with the new primary Node it issues a `ResendRequest` of previously processed messages (on the old primary) due to the most recently received FIX message sequence numbers not being replicated to the new primary at the point of failover. Most FIX applications should be able to handle this because such resent messages always have the `PosDupFlag(43)` tag set to `Y` indicating the resend.
- Should counterparties send a `ResendRequest` to the new primary it is possible the request cannot be serviced due to the most recently sent FIX messages not being replicated to the new primary at the point of failover.
- Persistent collections may be out of date due to the most recent write requests not being replicated to the new primary at the point of failover.

So there is a trade-off between performance when not using the write quorum versus the integrity of the data when it is in use. The choice of which is more suitable is likely to be environment specific.

### 6.1.5. Single IP Address

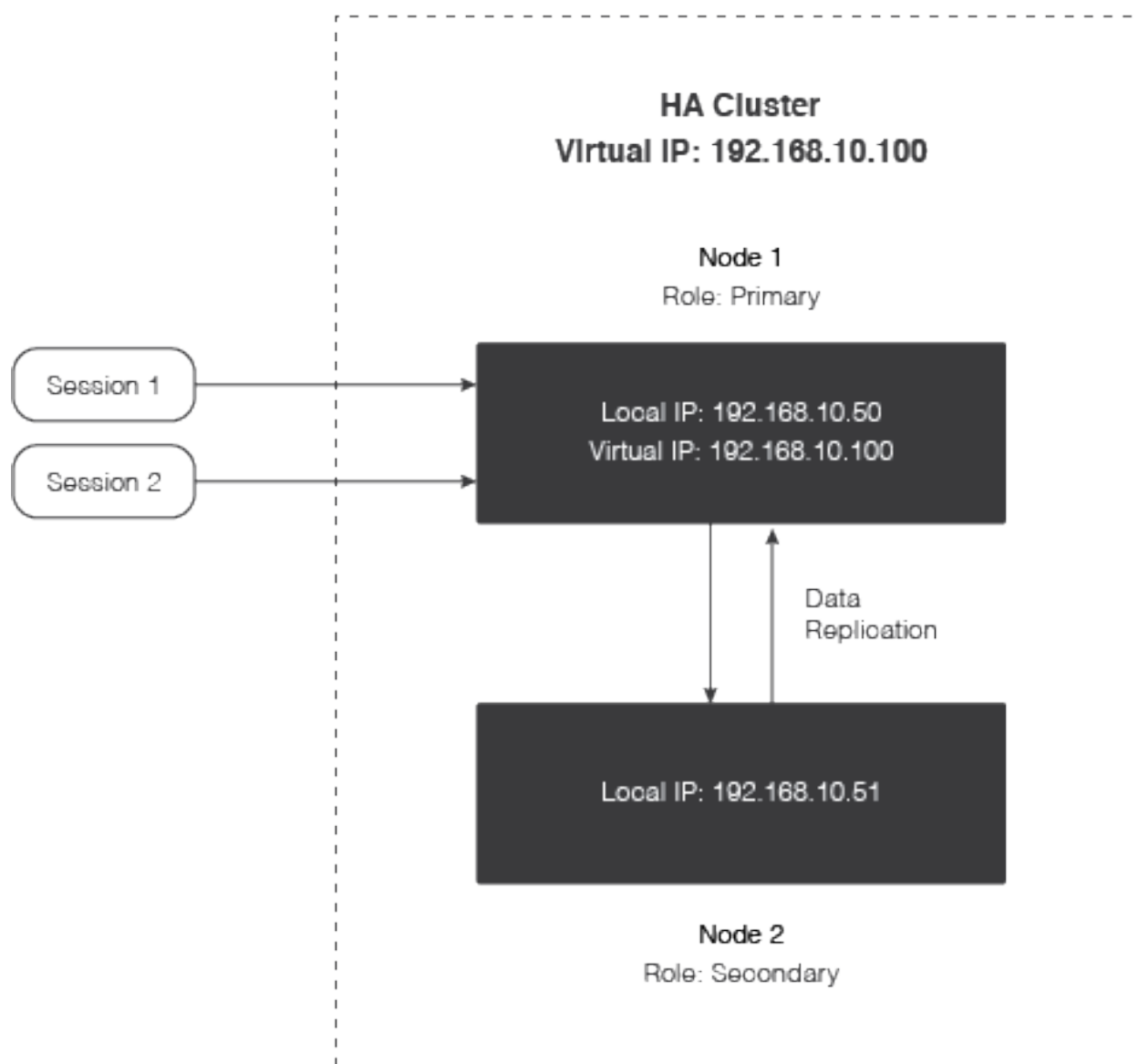
It is generally desirable for a clustered FIX service to present a single IP address to initiating counterparties to simplify their connection strategy. As mentioned earlier, there are two main methods of achieving this:

1. Use of a single virtual IP address (VIP) together with the use of gratuitous ARP (GARP) requests to force synchronization of upstream ARP tables.
2. Use of an upstream network device (e.g. content switch or load balancer) - generally itself in a redundant configuration - to determine which of the available Nodes is currently the primary and direct IP traffic accordingly.

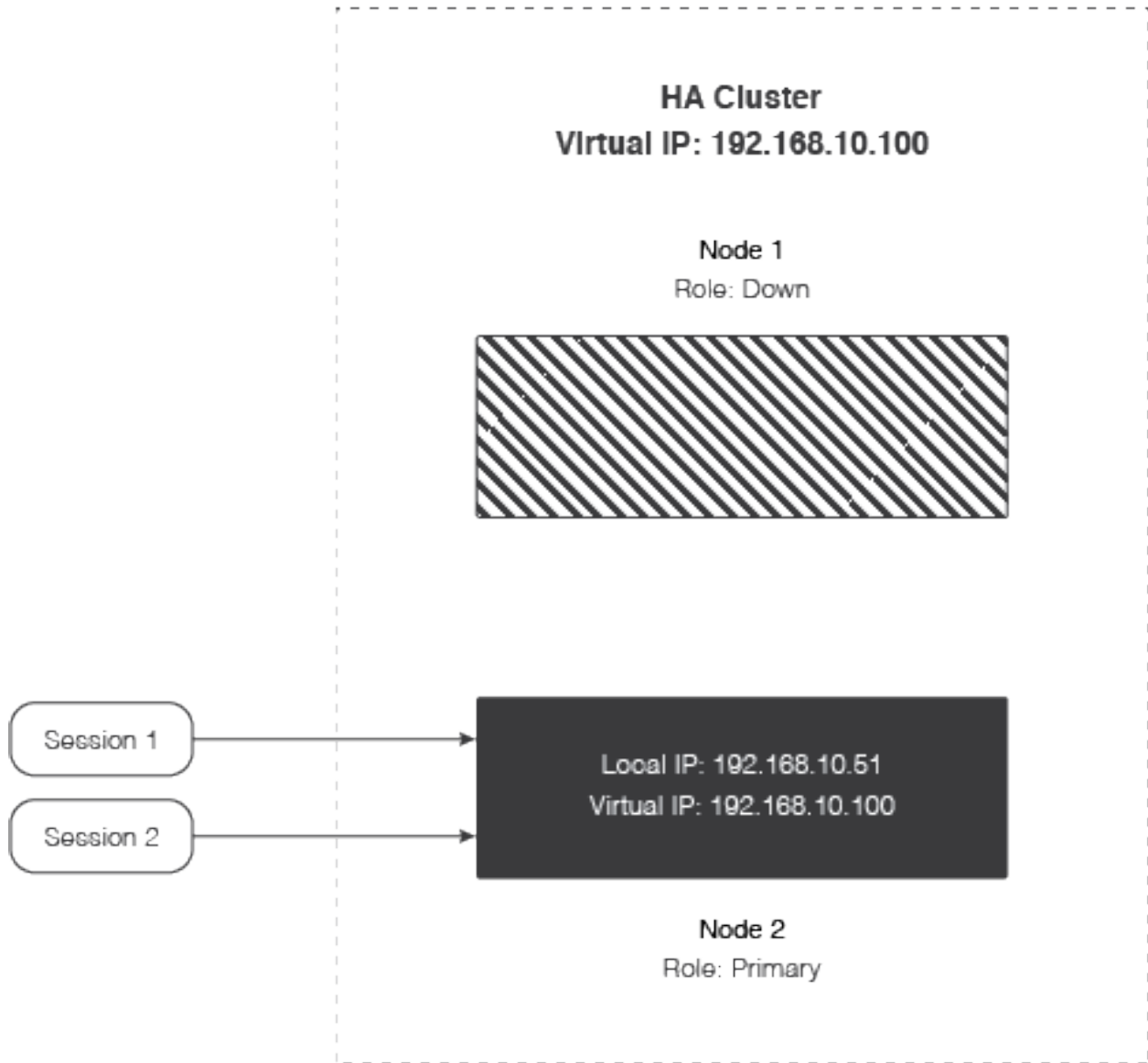
### IP takeover and Gratuitous ARP

The first method listed above is implemented using operating system level scripts which are triggered by the HA software at various phases of its role determination lifecycle. Working example scripts of this method on major operating systems are included in the Node distribution under *templates/HighAvailability/FailoverScripts*.

In the following example, Node 1 starts out in the primary role with Node 2 in a secondary role. It is configured with two IP addresses: the *local* address (192.168.10.50) and the *cluster* address (192.168.10.100), also known as the virtual IP (VIP). The counterparties communicate with this FIX provider via the VIP.



Upon failure of the Node 1, Node 2 will takeover the primary role and assume ownership of the VIP. The `vip-up.sh` (UNIX) or `vip-up.bat` (Windows) script on Node 2 is executed and sends a GARP request to force synchronization of upstream ARP tables. After this, all message traffic routed to the VIP will be sent to Node 2. As a result of the failover, the counterparty will experience a socket disconnection followed by a reconnection and FIX Logon exchange.



## Upstream Network Device

When an intelligent network device exists, often a load balancer, it is possible for it to determine which of the available Nodes is the current primary and route message traffic accordingly. This is made possible by a fundamental property of the Node lifecycle that guarantees that only components on a primary Node reach their active states. Identical components on the secondary Nodes remain inactive. Similarly, all active components on a primary are deactivated if a manual failover to another Node is initiated.

In the following example, Node 1 starts out in the primary role with Node 2 in a secondary role. The counterparties communicate with this FIX provider via the IP of the network device. The network device routes traffic to the Node in the primary role. The network device periodically checks the status of the primary. Some ways to accomplish this are: 1. Pinging a port that the primary Node activates. 2. Checking that the primary's process is running. 3. Custom implementation.



Upon failure of the Node 1, Node 2 will takeover the primary role and assume ownership of the VIP. The `vip-up.sh` (UNIX) or `vip-up.bat` (Windows) script on Node 2 is executed and sends a GARP request to force synchronization of upstream ARP tables. After this, all message traffic is routed to the VIP will be sent to Node 2. As a result of the failover, the counterparty will experience a socket disconnection followed by a reconnection and FIX Logon exchange.

To take advantage of this property, a typical solution is to:

- Code a simple [IService](#) implementation that is also a [ComponentHelper](#) which only listens (and potentially responds to requests) on a well known port, but ONLY when it is active
- Configure the service as a `GenericService`

In this scenario the network device may locate the current primary by simply determining the Node on which the service is active. Further, it can repeat the process periodically in order to detect failure of the current primary.

This, or a similar, technique may be used to have the Node inter-operate with existing clustering environments or third-party clustering implementations.

### 6.1.6. Configuration

It is highly recommended that all Nodes in an HA cluster have identical configurations. When a cluster member starts up, it compares its configuration with the other members and will abort if a mismatch is detected. If the `allowConfigMismatch` attribute on the `HighAvailabilityCluster` element is set to `true`, then differences in the non-HA sections of the configuration will be allowed. This option should only be used during testing or in other rare circumstances.



#### Note

Due to significant differences in operating systems, Catalys Node doesn't support clusters which members are running on machines with different operating systems. In particular, setup in which some part of the cluster runs on Windows and other part on other OS is not allowed.

If the configuration of a primary Node is modified at runtime (either via Dashboard, JMX or by reloading its modified XML configuration file), then the secondary Nodes automatically synchronize their configuration with that of the primary. If any secondary Node is down while runtime changes are made to the primary, then these changes need to be made manually on the secondary before starting it.



## Note

The only type of persister that can be used by FIX sessions in an HA cluster is the `JournalingPersister`. See [Session Persistence](#) for configuration details.

### 6.1.6.1. Configuration Example and Element Definitions

The following example shows an HA configuration with two Nodes at a primary site and one at a disaster recovery site. This configuration is targeted for a Unix platform — the specified shell scripts perform the ping test and manage the VIP. Sample scripts for Unix and Windows are included in the *templates/HighAvailability/FailoverScripts* directory of the Node distribution.

```
<Config>
  <Application id="BROKERFIX1">
    <HighAvailabilityCluster id="BROKERFIX1_CLUSTER">
      <ClusterSite id="PRIMARY">
        <ClusterNode id="HUB1" dataVersionId="10" host="192.168.10.50" port="9001"/>
        <ClusterNode id="HUB2" dataVersionId="20" host="192.168.10.51" port="9002"/>
        <WriteQuorumSpecification
          writeQuorumAtLocalSite="1"
          writeQuorumAtRemoteSites="0"
          writeQuorumAtAllSites="0"/>
      </ClusterSite>
      <ClusterSite id="DR">
        <ClusterNode id="HUB3" dataVersionId="30" host="192.168.11.1" port="9003"/>
        <WriteQuorumSpecification
          writeQuorumAtLocalSite="0"
          writeQuorumAtRemoteSites="0"
          writeQuorumAtAllSites="0"/>
      </ClusterSite>
      <PrimaryValidityChecks>
        <PrimaryValidityCheckScript id="pingtest" command="/home/cameron/pingtest.sh"/>
      </PrimaryValidityChecks>
      <BecomingPrimaryActions>
        <BecomingPrimaryScript id="master" command="/home/cameron/vip-up.sh"/>
      </BecomingPrimaryActions>
      <RelinquishingPrimaryActions>
        <RelinquishingPrimaryScript id="slave" command="/home/cameron/vip-down.sh"/>
      </RelinquishingPrimaryActions>
    </HighAvailabilityCluster>
    <Sessions>
      ...
    </Sessions>
  </Application>
</Config>
```

Each of the configuration elements is explained in detail below.

## HighAvailabilityCluster Element

All HA-related configuration is contained under this element. There can only be one of these present per Node instance. You configure the overall settings for your HA cluster on this element. For a complete list of attributes, see the [HighAvailabilityCluster Configuration Reference](#).



### Important

If you are required to handle large messages in your HA cluster, you must increase the `maxClusterMessageSize` attribute on the `HighAvailabilityCluster` element greater than the size of the largest message. If you attempt to handle a message larger than this value, then you will see `DistributedByteArrayStore` warnings in your log file and the data will not be replicated.

## ClusterSite Element

This element defines a group of Nodes that reside in the same logical site, which usually corresponds to a physical location, e.g. "Data Center #1". For a complete list of attributes, see the [ClusterSite Configuration Reference](#).

## ClusterNode Element

Each Node in an HA cluster is defined by a `ClusterNode` element. The `host` and `port` attributes represent the network address that other cluster members use to communicate with this Node (these are not necessarily the same IP addresses used for FIX traffic). Automatic failover and other network binding options can also be configured. For a complete list of attributes, see the [ClusterNode Configuration Reference](#).



### Note

The `dataVersionId` attribute is required and must be a unique integer value for each `ClusterNode` element. The value cannot be modified in between FIX session resets.

## WriteQuorumSpecification Element

Write quorum defines the number of acknowledgments the primary Node must receive in order for a persistence update to be considered replicated. There are 3 different write quorum settings for each `ClusterSite` element: local site, remote site and all sites.

For a complete list of attributes, see the [WriteQuorumSpecification Configuration Reference](#).

## PrimaryValidityChecks Element

The `PrimaryValidityChecks` element, which has no attributes, contains a list of one or more checks that a Node executes before it can declare itself the primary Node of a cluster. If any of the checks fail, then it will not declare itself primary.



### Important

The minimum recommended check is to ping one or several devices on the network, e.g. the default router and/or a device on the other side of the default router. Pinging one of the other members in the cluster is **not** recommended, since this may give a false indication of the Node's network status. Checks are implemented as shell scripts or batch files and are referenced via the `PrimaryValidityCheckScript`.

## PrimaryValidityCheckScript Element

The `PrimaryValidityCheckScript` element refers to a shell script or batch file that is executed before the Node attempts to become primary. The script must return 0 if successful. If any of the configured scripts return a non-zero value, then the Node will not declare itself as the primary in the cluster. At least one check must be configured.

For a complete list of attributes, see the [PrimaryValidityCheckScript Configuration Reference](#).

## Ping Primary Validity Check Element

One of the primary validity checks that can be configured is a ping command. This command pings one or more IP addresses to make sure that the Node is still connected to the network. If the server fails to successfully ping any of the addresses it will not declare itself primary.

The `Ping` element can contain one or more `Destination` elements which contain a single `host` attribute whose value is the IP or host of the server to ping. For a complete list of attributes, see the [Ping Configuration Reference](#).

```
<PrimaryValidityChecks>
  <Ping id="ping">
    <Destination host="192.168.100.1" />
  </Ping>
</PrimaryValidityChecks>
```

There are some [limitations](#) to consider when using the `Ping` check.

## BecomingPrimaryActions Element

Once a Node has successfully passed all of its primary validity checks, it has the option of executing a series of actions. Most commonly this is used to take control of the cluster's virtual IP address.

This element has no attributes.

## BecomingPrimaryScript Element

The `BecomingPrimaryScript` element refers to a shell script or batch file that is executed when the Node becomes primary. The script must return 0 if successful. If any of the configured scripts return a non-zero value, then the Node will not declare itself as the primary.

For a complete list of attributes, see the [BecomingPrimaryScript Configuration Reference](#).

## RelinquishingPrimaryActions Element

When a Node is changing its role from primary to secondary, in addition to the relinquishing actions the node will perform, it has the option of executing a series of custom actions. Most commonly this is used to relinquish control of the cluster's virtual IP address.

For configurations and operating systems where IP takeover cannot occur without the IP address being explicitly relinquished, the LMA may need to run the Relinquishing Primary Actions. For more details, see [Using the LMA to Relinquish Primary Actions](#).

In the scenario where the host on which the Node is running goes down and comes up again, the Node's Relinquishing Primary Actions script will not run. This will mean that the IP address of the cluster is not explicitly relinquished, and in some cases on some operating systems, the cluster IP address cannot be acquired by the new primary.

In some cases it may be necessary to run the Relinquishing Primary Actions at system startup. This ensures that the network remains in a consistent state in the situation where the hardware on which the host is running goes down and comes back up again. On Unix this can be achieved using an *init.d* entry, and on Windows by using a service.

## RelinquishingPrimaryScript Element

The `RelinquishingPrimaryScript` element refers to a shell script or batch file that is executed when the Node transitions from primary to secondary. The script must return 0 if successful. If any of the configured scripts return a non-zero value, the Node will proceed with relinquishing its role as primary but will subsequently shut down.

For a complete list of attributes, see the [RelinquishingPrimaryScript Configuration Reference](#).

## Using the LMA to Relinquish Primary Actions

In the scenario where a Node has become unresponsive but is still running, its Relinquishing Primary Actions script may not run. This may mean that the IP address of the cluster is not relinquished, and in some cases on some operating systems, the virtual IP of the cannot be acquired by the new primary.

In this case the LMA can be used to run the Relinquishing Primary Actions script on the old primary host, as well as kill the unresponsive application.

To do this, configure a custom `pgrep` command on the LMA, which includes both finding the primary application and relinquishing the IP address. That is, append the commands that relinquish the IP address to the standard `pgrep` command.

To configure a custom `pgrep` command, refer to "Specifying Custom `pgrep` and `pkill` Commands" in the Configuration chapter of the LMA documentation.

## 6.1.7. High Availability Operations

### 6.1.7.1. Starting an HA Node

When starting up a Node that is part of an HA cluster, there is an additional required application argument, `-clusterNode`, which identifies the Node in the cluster that is being started. The value of the argument must match the `id` attribute of one of the `ClusterNode` elements.

```
java -cp "../lib/catalys-node.jar:../resources" com.camerontec.catalys.server.CatalysServer \
  -xmlconfig config.xml -clusterNode HUB1
```

### 6.1.7.2. Write Quorum Settings

The number of configured acknowledgements must be strictly less than the number of Nodes in the cluster (or less than or equal to the number of secondary Nodes in the cluster, since there is a maximum of one acknowledgement per secondary Node). The number of configured acknowledgements is the sum of all of the writeQuorums. That is:  $\text{writeQuorumAtLocalSite} + \text{writeQuorumAtRemoteSites} + \text{writeQuorumAtAllSites} < N$  (where  $N$  is the number of Nodes in the cluster).

The number of required acknowledgements depends on how many secondary Nodes are actually up and running in the cluster. The maximum number of required acknowledgements is equal to the number of secondary Nodes present. If some of the secondary Nodes are not running then they cannot count towards the quorum, and their contribution is thus removed from their quorum specification (whether they are local or remote).

The `writeQuorumAtAllSites` value allows the acknowledgement to come from either the local or remote site(s).

In general, the following guidelines can be used:

- If there is no requirement for acknowledgements, set all three write quorum attributes to 0.
- To require acknowledgements from all secondary Nodes, set `writeQuorumAtLocalSite` to one less than the number of local Nodes, `writeQuorumAtRemoteSites` to the number of remote Nodes, and `writeQuorumAtAllSites` to 0.

## Optional Features

- To require any one Node to acknowledge, set `writeQuorumAtLocalSite` and `writeQuorumAtRemoteSites` to 0, and set `writeQuorumAtAllSites` to 1.

When there are two Nodes in the same `ClusterSite`:

- If there is no requirement for acknowledgements, set all three write quorum attributes to 0.
- Otherwise, set either `writeQuorumAtLocalSite` or `writeQuorumAtAllSites` to 1.

When there are two Nodes in one `ClusterSite` and one Node in another `ClusterSite`:

- If there is no requirement for acknowledgements, set all three write quorum attributes to 0.
- If data safety is more important than performance, set `writeQuorumAtLocalSite` and `writeQuorumAtRemoteSites` to 0 and `writeQuorumAtAllSites` to 1. If the local secondary goes down, the primary will wait for acks from the remote Node.
- If performance is more important than data safety, set `writeQuorumAtLocalSite` to 1 and `writeQuorumAtRemoteSites` and `writeQuorumAtAllSites` to 0. If the local secondary goes down, the primary will not wait for acks from the remote Node
- If there is a requirement for two acknowledgements and there are two secondary Nodes available, set the three write quorum attributes such that the values add up to 2.

### 6.1.7.3. Sequence Numbers on Primary and Secondary Nodes

The manner in which sequence numbers are reported differs on primary and secondary Nodes.

On the primary, the next expected sequence number is reported, whereas on secondary Nodes, the last processed sequence number is reported.

Consequently the reported sequence number on secondary Nodes can lag behind the reported sequence number on the primary. The amount by which it lags is determined by the number of messages yet to be processed. The lag is most commonly one.

### 6.1.7.4. Consequences of Timeout of Checks and Actions

If any of the timeouts (`primaryValidityChecksTimeout`, `becomingPrimaryActionsTimeout` or `relinquishingPrimaryActionsTimeout`) are exceeded, then the checks or actions that were being executed (`PrimaryValidityChecks`, `BecomingPrimaryActions`, `RelinquishingPrimaryActions`) are interrupted.

For actions that consist of running external commands, once the timeout is reached, the external process is forcibly killed and is assumed to have returned with an exit value of -1.

Actions implemented in Java are executed in a dedicated thread. Once the timeout is reached, this thread is interrupted (through a call to `Thread.interrupt()`). It is therefore important that the action's implementation handles `InterruptedException` properly (i.e. stops executing as soon as an `InterruptedException` is caught). If an action's implementation ignores such interruptions (typically by

catching and swallowing), the action's executing thread will keep running forever, which will effectively prevent the server from being able to change its role again later (because action threads are obtained from a thread pool of size one).

### 6.1.7.5. Limitations of the Built-In Ping Validity Check

The [Ping](#) validity check is implemented using Java's [InetAddress.isReachable\(\)](#) method which has a number of limitations due to Java's lack of built-in support for raw sockets (ICMP is layered on top of IP, whereas Java only provides TCP or UDP sockets).

For that reason, this implementation does its best effort to make use of the platform's built-in `ping` command/library, if one is available, and if sufficient privileges can be obtained for that.

If such a library is not available, the implementation resorts to sending a message to the TCP Echo service (port 7) on the specified host and seeing if a response comes back. This is very different from actual ICMP-based ping because the Echo service typically only runs on Unix workstations and servers, not on routers or switches.

The situation varies with the platform the JVM is running on:

- On Unix, the implementation tries to execute the system's `ping` command, which will only succeed if the JVM is run as root (because `ping` requires root privileges). If the JVM is not run as root, then a request is sent to the TCP Echo service.
- On Windows, the implementation does not make use of the platform's ping DLL and therefore always sends a request to the TCP Echo service.

In practice, it means that if the destination host of the `ping` validity check does not have a TCP Echo service running, the check will always fail unless the JVM is running as root on a Unix system.

For all those cases where the built-in HA ping cannot be used, there are two alternatives:

- Use a [PrimaryValidityCheckScript](#) element instead, and write a shell script or batch file that makes use of the system's `ping` command.
- Write a custom implementation in Java and specify it as a [GenericPrimaryValidityCheck](#) in the configuration.

### 6.1.7.6. Console Commands

The following HA commands can be issued from the [Command Line](#).

Command	Description
<code>cl_list</code>	List all the Nodes in the HA cluster and their current state.
<code>cl_primary</code>	Force the current Node to become the primary in the HA cluster. This will first force the current primary to release the primary lock.



## Optional Features

Command	Description
<code>cl_shutdown &lt;true false&gt;</code>	This command with <code>true</code> , only shuts down the local Node, with <code>false</code> it shuts down all Nodes in the HA cluster. Defaults to <code>false</code> .
<code>cl_replay_on</code>	<p>Force the current Node to process replay requests from other Nodes, regardless of configuration. This command can be used on a disaster recovery (DR) Node after manually failing over to that DR Node, and some processing happened. A DR Node is typically configured not to process replay requests from other Nodes by default, which means that attempting to bring any other Node online will fail because it cannot synchronize. This command provides a way to temporarily override the DR's replay response processing state to allow other Nodes to be brought back online in a controlled manner.</p> <p>This command must be used with care, since it will cause data on other cluster Nodes to be replaced by that of the local Node.</p> <p>Note that the replay request processing override state is not persisted and not replicated to other Nodes. If the Node is restarted its configured state will be re-instated.</p>
<code>cl_replay_off</code>	<p>Prevent the current Node from processing replay requests from other Nodes, regardless of configuration. This command is typically used on a disaster recovery (DR) site to restore the default behaviour after using the <code>cl_replay_on</code> command to allow other Nodes to be brought back online.</p> <p>Note that the replay request processing override state is not persisted and not replicated to other Nodes. If the Node is restarted its configured state will be re-instated.</p>
<code>cl_sync_win &lt;window size&gt;</code>	<p>Set the synchronization window on this Node to the specified size. A size of 0 means an unlimited size, which implies there is no limit placed on the size of the synchronization data set.</p> <p>Note that the new synchronization window size is not persisted and not replicated to other Nodes. If the Node is restarted its configured state and window size will be re-instated.</p>
<code>cl_restore_defs</code>	Restores the default (i.e. configured) values for the replay processing state and the synchronization window. After making manual changes, it is recommended to use this command to restore the defaults. A Node restart has the same effect.

### 6.1.7.7. Using the Local Management Agent (LMA)

In High Availability configurations, the LMA is utilized to ensure a greater level of reliability.

During failover, the Node that is the new primary candidate confirms that the previous primary is not running. If it is running, but has become unresponsive, the process is killed by the LMA. This is referred to as *LMA-kill*.

In addition, if the `becomePrimaryWithoutLMAConfirmation` attribute of the `HighAvailabilityCluster` element is set to `false` but the LMA cannot be contacted or cannot kill the application, then the failover will not succeed. This guards against the split-brain scenario (more than one Node becoming primary) in an unreliable network.



#### Note

To utilize the LMA-kill feature, the LMA must be at version 2.0 or later.

### Application ID Attribute

In order for the LMA-kill feature to work, all applications in a cluster need to have the same ID. At startup, the local application ID is compared with those of remote cluster Nodes and a warning is given if any difference is found. This is necessary because secondary Nodes must be able to construct the JMX object name of the application MBean exposed by the primary LMA (the MBean on which the `kill` operation will be invoked). This MBean's name contains the application ID.

### PID Registration

An additional requirement for this feature to work is that each server must register its PID (process ID) with the LMA. This is done by default if the server is running on Unix, or on Windows with an Oracle JVM. If the operating system is Windows and the JVM is not Oracle, a custom command to retrieve the PID is always required.

This is configured using `getPIDCommand` attribute of the `JMXManagementService`.

The recommended implementation of the `getPIDCommand` is a small utility called `getpids.exe` that is [freely available for download](#). This utility is not included in the Node distribution.

Below is a configuration extract that shows how to pass a custom command that makes use of this utility to the `JMXManagementService`. It is assumed that `getpids.exe` is in the `%Path%`.

```
<Services>
  <GenericService
    class="com.camerontec.catalys.server.management.JMXManagementService"
    id="JMXManagementService">
```

```
<Properties id="JMXManagementServiceProperties">
  <Property
    name="getPIDCommand"
    value="for /F &quot;usebackq tokens=2&quot; %p in (`getpids.exe`); do @echo %p" />
  </Property>
</GenericService>
</Services>
```

### Custom LMA Commands

It is also possible to alter the LMA's configuration in order to specify custom commands for locating and killing processes. For most scenarios this is not recommended, but one situation where this might be necessary is described in [Using the LMA to Relinquish Primary Actions](#).

#### 6.1.7.8. Known Issues

A secondary Node trying to synchronize when the primary is under a very heavy load can potentially run out of memory. When synchronizing against the rest of the cluster, the Node buffers replicated writes coming from the primary in an in-memory queue. Once synchronization is done, the contents of that queue are replayed.

Although data synchronization was designed to be very fast, there are cases where it can take a long time to complete. For example, adding a new Node with empty persistence to a cluster that has already processed a lot of data will result in a long synchronization. This should be avoided in general, because doing this will likely impact the performance of the cluster. There are however some situations where it may be desirable to do it.

Because the in-memory queue of replicated writes is not bounded, it will keep growing for the entire duration of the synchronization. Depending on the kind of traffic that the primary is processing when that happens, it is possible that this queue causes the late-joining cluster Node to run out of memory.

Should this happen, the recommended approach is to stop the late-joining cluster Node and to only attempt to bring it on-line when the primary is under a more reasonable load.

### 6.1.8. High Availability Communications

#### 6.1.8.1. Introduction

The High Availability (HA) communications subsystem is at the base of the HA stack and therefore forms the basis on which the remainder of the solution is built. In order to configure the cluster for your particular environment it is important to understand how the communications subsystem operates. This section is designed to provide the required understanding and guidance towards the configuration appropriate for your environment and cluster characteristics.

#### 6.1.8.2. Context

The internal HA services layered on top of the communications system are:

- **Group Membership Protocol:** The group membership protocol allows each Node to broadcast its presence and its current role. The protocol is implemented as a periodic broadcast of local state (here the term *broadcast* simply means that each Node sends its role state to all other Nodes - no particular underlying transport is implied). The periodicity is low so that bandwidth requirements are minimal.
- **HA Control Operations:** There are many operations which occur at startup, on demand and when certain error conditions are detected. Such operations are typically point to point and once the system is in steady state do not occur very often. Bandwidth requirements are low.
- **Data Replication:** Data written to any distributed collection (i.e. any collection obtained via the [CollectionRegistryService](#) when running in HA mode) is both persisted to the local disk and replicated to other cluster Nodes. The frequency of updates is directly proportional to the total number of collection updates, which can be very high in a loaded system. Therefore, the bandwidth requirements of this service can be very high. In addition, observed application latency depends on data replication and acknowledgement (if required) latency so it is important that this is minimized.
- **Data Synchronization:** Whenever a Node joins the cluster or a distributed collection is detectably out of sync with the master Node, data synchronization occurs. A dedicated (distributed) state machine exists per collection to control this and in particular determine the most efficient means of synchronization. The goal is always that synchronization time is proportional to the outage time (i.e. the number of missed writes) rather than the size of the collection itself. When the underlying network is solid, this does not happen very often. When it does happen, however, it is potentially disruptive to the running cluster especially if it is under high load and the Node being synchronized as a long way out of date. Traffic of this type therefore typically occurs in bursts.

There are therefore two primary classes of traffic with quite different characteristics:

1. Control traffic: low bandwidth, low periodic or demand based
2. Data traffic: high bandwidth, high frequency

The HA communications subsystem is therefore designed to take these different types of traffic into consideration while at the same time being scalable to a reasonable number of cluster Nodes.

### 6.1.8.3. Discovery

Before the communications subsystem is ready to support HA services it must first establish communications channels with all other cluster Nodes. When a particular cluster Node starts up it has access to the HA configuration and therefore knows which other Nodes it should be able to find. It cannot, however, make any assumptions about which other HA Nodes are actually running. The first step in the discovery process, therefore, is for each cluster Node to initiate a TCP connection with every other configured cluster Node. In the normal case there are two TCP connections between each cluster Node (i.e. one initiated by each Node). We know there are two primary classes of traffic to support, so both TCP connections are retained and each assigned to a designated traffic class. This process is known as channel role assignment and a simple, deterministic, handshake occurs in order to facilitate it. Once this has occurred the cluster Nodes regard both channels as being available for use by higher level services.

All messages thereafter are sent on the appropriate channel for their traffic class. The mapping of which messages are sent over which channel is the domain of the communications subsystem itself. The rule is very simple though: only data writes and synchronization requests and responses are sent over the data channel. Everything else is sent over the control channel. It should never be the case that a control message suffers process delay due to high data throughput because this has a potentially destabilizing effect on the cluster itself.

Using two TCP connections to carry all HA communications traffic is the default situation. As is discussed in the [Multicast-based Data Replication](#) section below, it is also possible to enable multicast for data synchronization. Whether this is the best option depends a lot on the number of Nodes in the cluster, whether datagrams propagate to all Nodes, the path MTU and the quality of the underlying network.

### 6.1.8.4. Stacks

The HA communications subsystem consists of two, logically distinct, asynchronous, bi-directional stacks:

1. Control
2. Data

The outbound path of each stack has the following (simplified view of) components:

- Higher layer threads (of which there could be many) sending messages. Serialization occurs on the sender's thread.
- A buffer for storing serialized outbound messages.
- A single buffer consumer thread for writing serialized message to the appropriate channels. Depending on which transports are enabled, multiple writes may be required for a single message (e.g. writing data messages to multiple slave Nodes via TCP).

The inbound path of each stack has the following (simplified view of) components:

- A single thread which handles all channel reads for a particular channel role. That is, reading from all control or data channels associated with all other Nodes. Note that this thread, when required, is also responsible for sending write request acknowledgement messages.
- A buffer for storing serialized inbound messages.
- A single buffer consumer thread for notifying channel subscribers (which are normally higher layer services) of inbound messages. Each subscriber is responsible for deserializing its own messages.

With the design implied above, it is clear that the threading model is fixed regardless of the number of Nodes in the cluster. Specifically, we have:

- Two writer threads (one for each of Control and Data stacks)
- Two reader threads (one for each of Control and Data stacks)
- Two notifier threads (one for each of Control and Data stacks)

In addition there is another slow periodic controller thread which is responsible for channel health monitoring and recovery. Therefore we have seven threads in total performing all HA communications subsystem work, however, not all threads are equal. In particular, observed application performance is likely to be maximized when:

- The data writer buffer is sufficiently large so that higher layer writer threads are never blocked awaiting free space.
- The data writer thread is configured with an appropriate buffer wait strategy for the expected throughput. A blocking strategy may be acceptable if there is always expected to be some load on the system, however, to try to guard against context switch latency after quiet periods a busy/spin strategy is more appropriate. If a busy/spin buffer wait strategy is used, the data writer thread should also be bound to an isolated CPU core.
- The data reader thread is configured with an appropriate I/O multiplexing strategy. Again, a blocking strategy may be acceptable if there is always expected to be some load on the system, however, to try to guard against context switch latency after quiet periods, a busy/spin strategy is more appropriate. If a busy/spin I/O multiplexing strategy is used, the data reader thread should also be bound to an isolated CPU core.
- The data notification buffer is sufficiently large so that the data reader thread is never forced to wait for a free slot. Note that in steady state the data notification buffer need only be as large as the data write buffer, however, to also cope with bursts associated with (unexpected) data synchronization it is recommended to allocate a much larger data notification buffer. The default is two orders of magnitude larger.

Obviously data serialization costs are also extremely relevant to observed application performance, however, they are not considered to be part of the communications subsystem so not directly relevant to this section.

### 6.1.8.5. Multicast-based Data Replication

Where the environment permits it is possible to configure the system to perform data replication over multicast. In order to do this it is necessary to configure both the [multicastAddress](#) and [multicastPort](#) on the [HighAvailabilityCluster](#) element.

There are a number of additional considerations which become important in this case:

1. Do datagram packets propagate to all cluster Nodes or only Nodes in the main site?
2. If datagram packets **do not** propagate to Nodes outside the main site, then how many such Nodes exist?
3. Is the [write quorum specification](#) in the main site being used (i.e. are any of them non-zero)?
4. What is the path MTU between the primary Node and all other Nodes which can receive datagram packets?

If the network permits datagrams to Nodes outside the primary site it may be necessary to configure the [multicastTimeToLive](#) attribute such that the packet does not expire before they reach the desired Nodes.

It is still possible to use multicast based replication if datagram packets do not propagate to non-primary site Nodes by enabling [passive site replication](#). In this mode all Nodes outside the primary site periodically request synchronization from the lowest priority Node in the primary site. This places a lot of additional load on that Node and so the effectiveness of this approach depends on how many such Nodes exist. Ultimately, if there are many Nodes outside the primary site which do not receive datagram packets and the system sustains high throughput, multicast may not be the best option.

Explicit data write request acknowledgements are still sent (and expected to be sent) whenever the effective write quorum is non-zero. Acknowledgements to write requests received over multicast are sent to the originator over unicast. Network load is still reduced, however, in general multicast works best when write acknowledgements are disabled.

Neither fragmentation nor packet loss nor any other issues inherently associated with datagram based traffic are handled explicitly. Rather the following trade-offs are in place:

- In order to avoid packet fragmentation and associated issues, the system is configured in advance with the maximum allowed MTU. Whenever the length of a data write request exceeds this it is sent over TCP instead of multicast. Fortunately, modern hardware typically supports path MTUs which make typical data packets (e.g. FIX messages) less likely to be fragmented. It is recommended to measure the path MTU and configure the [maxDatagramSize](#) accordingly.
- Any transport based on UDP is subject to packet loss. Apart from the quality of the network, the most important parameter which can be used to minimize packet loss is the receiver's [udpReceiveBufferSize](#). This should be configured to be as large as the host OS permits. Note that it may be necessary to tune the host OS to allow large buffers. The system will not start unless the OS actually allocates a buffer of the requested size.
- Although the HA communications subsystem itself does not support retries for lost datagrams, this is handled at a higher layer by each distributed collection's state machine. That is, a single lost packet at the level of the communications subsystem only affects a single collection and not all subsequent packets. The state machine then takes steps to achieve distributed collection synchronization via the most efficient means. Another advantage of this approach is that the same detection and correction scheme is used regardless of the underlying transport.

## 6.2. Catalys Rules Engine

### 6.2.1. Introduction

The Catalys Rules Engine (CRE) provides built-in functionality to interact with inbound and outbound FIX messages on any session and does not require writing Java code. The CRE is typically used for these purposes:

- **Enrichment:** Adding, removing, copying, converting or replacing FIX fields and their values.
- **Routing:** Routing messages from one session to another or drop copying messages to one or more sessions. Also the ability to route messages to different technology adapters.
- **Validation:** Ensuring that inbound messages contain or do not contain particular tags and/or values or that a field's value is unique on a session, and giving the option of rejecting the message if it is invalid.
- **Store/Restore:** Storing FIX fields to be restored on a related message.

In addition, if there is desired functionality that is not supported by the CRE, an API is provided that allows custom conditions and actions to be developed. See [GenericCondition](#) and [GenericAction](#).

## 6.2.2. Configuration

Rules are grouped into one or more *rules packs*, which are defined at the application scope. Individual sessions can then reference rules packs by ID. Multiple sessions can reference the same rules pack.

A rule is made up of *conditions* and *actions*; each action is performed on the message if the conditions are satisfied. Rules are executed in the order they are listed in rules pack. Certain actions (reject, route, discard) are considered *final*; after such an action is executed, the message does not pass through any subsequently defined rules.

It is possible to add or modify rules at runtime via the Dashboard, the JMX API or by manually editing the XML configuration file. Care needs to be taken when changing rules dynamically, especially when changing rules which have inbound and outbound interactions, for example a response to a message created by the original set of rules may be received by the new set of rules.

### 6.2.2.1. Configuration Example

In example below, the `TimeInForce(59)` field is added to all outbound order messages, and the `Account(1)` and `TimeInForce(59)` fields are removed from incoming execution reports:

```
<Config>
  <Application id="FIX-INSTANCE1">
    <!-- the rules pack defining rules -->
    <RulesPack id="order-rules">
      <!-- the rule defining conditions and actions -->
      <Rule direction="outbound" description="Outbound Order Rules">
        <Conditions>
          <!-- condition which matches order messages -->
          <Inclusions>
            <ConditionMatchMsgType values="D,G,F" />
          </Inclusions>
          <Exclusions/>
        </Conditions>
        <Actions>
          <!-- action which adds tag 59 with value "1" to the message -->
          <ActionAddTags tags="59" value="1" />
        </Actions>
      </Rule>
    </RulesPack>
  </Application>
</Config>
```



## Optional Features

```
<Rule direction="inbound" description="Inbound Exec Rules">
  <Conditions>
    <!-- condition which matches exec report messages -->
    <Inclusions>
      <ConditionMatchMsgType values="8" />
    </Inclusions>
    <Exclusions/>
  </Conditions>
  <Actions>
    <!-- action which removes tag1 and 58 from the message -->
    <ActionRemoveTags tags="1,58"/>
  </Actions>
</Rule>
</RulesPack>
<Sessions>
  <Session fixversion="4.4" compid="FUND" counterpartycompid="BROKER">
    <Connections>
      <SocketConnection id="CC1" port="2000" hostname="localhost" />
    </Connections>
    <SessionManager>
      <SourceMessageProcessors>
        <!-- Enable the rules engine on outbound messages -->
        <CatalysRulesEngine id="cre" rulesPackId="order-rules"/>
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <!-- Enable the rules engine on inbound messages -->
        <MessageProcessorReference id="cre_listener" refid="cre"/>
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
</Sessions>
</Application>
</Config>
```

Each of the configuration elements is explained in detail below.

### RulesPack Element

Contains a set of rules that typically perform a logical task. The `id` attribute is how sessions refer to a `RulesPack` element; the value must be unique across all other rules packs.

For a complete list of attributes, see the [RulesPack Configuration Reference](#).

### Rule Element

Contains `Condition` and `Action` elements. Normally a rule performs one operation on the message. Rules are executed in the order they are listed in the rules pack.

Attribute	Description
direction	The direction of the rule. Can be <code>inbound</code> , <code>outbound</code> , <code>biDirectional</code> , or <code>exception</code> . The <code>exception</code> direction is a special value which applies to inbound message processing only. When this value is set, exceptions

Attribute	Description
	thrown by any of the downstream processors are caught, and transformed into a message configured by the rule's actions. For example, in a routing scenario where downstream links are not up yet, exceptions which would normally result in Business Message Rejects can be transformed into Execution Reports with the appropriate Order Status value.  Required.
description	A textual description of the rule which is displayed in the Rules configuration GUI of the MIS/Dashboard.  Optional.
defaultCase	If <code>true</code> , this rule will be executed if no other previous rules have been executed. Note that no other rules will be evaluated after this rule is executed.  Optional. Defaults to <code>false</code> .
disabled	Whether this rule is currently active or not.  Optional. Defaults to <code>false</code> .

## Condition and Action Elements

Please refer to the [Conditions](#) and [Actions](#) sections.

## CatalysRulesEngine Element

The `CatalysRulesEngine` element is configured as part of a session's source and/or listener message processor chain. To use a single instance of the processor in both the source and listener chains, define the `CatalysRulesEngine` in one chain, and refer to it using the [MessageProcessorReference](#) element in the other.

For a complete list of attributes, see the [CatalysRulesEngine Configuration Reference](#).

## Tags Attribute

Many of the conditions and actions have a `tags` attribute. This consists of a comma- or space-delimited list of FIX field integers or names. For example, the following values are all valid:

- `tags="Symbol, OrderQty, Currency"`
- `tags="55 38 15"`
- `tags="55, OrderQty, 15"`

## 6.2.3. Expression Language

The `ConditionEvaluate` condition and `SetExpression` action support an expression language that is similar to JSTL. Fields can be referenced using FIX integer or name values, and basic logical and arithmetic operations are supported.

If specified using XML, quote characters in expressions must be escaped as `&quot;`.



### Tip

While quick to configure, expression evaluation is much slower than the equivalent `GenericAction` or `GenericCondition` implemented in Java. If performance is degraded due to use of expressions, consider replacing them with an equivalent custom or built-in action.

## General Examples

Expression	Result
1	1
tag35	The value of the MsgType(35) field
MsgType	The value of the MsgType(35) field
tag11==tag2011	True - if the values of the two fields are equal
tag11!=tag2011	True - if the values of the two fields are not equal
1+2	3
OrderQty/100	The value of the OrderQty(38) field divided by 100
tag38>100 && tag38<200	True - if the value OrderQty(38) is between 100 and 200
tags[78]	The value of the NoAllocs(78) field
fields[78]	The value of the NoAllocs(78) field
tags[78][1][79]	The value of the AllocAccount(79) field from the second repeating group entry. Note: repeating group entries are zero indexed based.

## Optional Features

Expression	Result
<code>fields[78][1][79]</code>	The value of the <code>AllocAccount(79)</code> field from the second repeating group entry. Note: repeating group entries are zero indexed based.
<code>tags[78][1][539][0][524]</code>	The value of the <code>NestedPartyID(524)</code> field from the first entry of the nested repeating group from the the second repeating group entry of the <code>AllocAccount(79)</code> field. Note: repeating group entries are zero indexed based.
<code>fields[78][1][539][0][524]</code>	The value of the <code>NestedPartyID(524)</code> field from the first entry of the nested repeating group from the the second repeating group entry of the <code>AllocAccount(79)</code> field. Note: repeating group entries are zero indexed based.
<code>isFieldPresent(TimelnForce)</code>	True - if the <code>TimelnForce(59)</code> field is present in the message.
<code>round(tag15, maxDecimals)</code>	Round the value of <code>Commission(15)</code> field up to the <code>maxDecimals</code> decimal amount. Trailing zeros are not stripped. Additionally, if the input value is zero it will create a decimal equivalent. An exception is thrown if the field's value is not a number.
<code>valueOf(tag31, maxDecimals)</code>	Round the value of <code>LastPx(31)</code> field up to the <code>maxDecimals</code> decimal amount. Trailing zeros are stripped. An exception is thrown if the field's value is not a number.

## String Functions

The following string manipulation functions are available - see the [java.lang.String](#) documentation for a full explanation of their function.

All string-based function arguments can be FIX message fields or a string in single quotes. Fields that are numbers will be converted to strings. Functions can be nested within each other, for instance:

```
concat(concat(toUpperCase(tag55), '.'), toUpperCase(tag65))
```

Returns:

```
Symbol(55) + '.' + SymbolSfx(65)
```

Function	Result
<code>concat('PRE', 'FIX')</code>	Returns 'PREFIX'

## Optional Features

Function	Result
concat(tag55, tag65)	Returns the combined value of the Symbol(55) and SymbolSfx(65) fields
substring(tagN, beginIndex, endIndex)	Returns the substring that begins at the specified beginIndex and extends to the character at index (endIndex-1) of field N or string
toUpperCase(tagN)	Returns the value of tagN to uppercase
toLowerCase(tagN)	Returns the value of tagN to lowercase
trim(tagN)	Returns the value of tagN with the white spaced trimmed from beginning and end
contains(tagN, string)	Returns true if tagN contains the value of string
startsWith(tagN, string)	Returns true if tagN starts with the value of string
endsWith(tagN, string)	Returns true if tagN ends with the value of string
matches(tagN, regex)	Returns true if tagN matches the value of string
indexOf(tagN, string)	Returns zero based index of the first occurrence of string in tagN
lastIndexOf(tagN, string)	Returns zero based index of the last occurrence of string in tagN
length(tagN)	Returns the length of the tagN
replaceAll(tagN, string1, string2)	Replaces all occurrences of string1 with string2 in tagN
replaceFirst(tagN, string1, string2)	Replaces the first occurrence of string1 with string2 in tagN

## Time and Date Functions

Function	Result
utcTimestampMs()	Returns the current GMT date and time in the yyyyMMdd-HH:mm:ss.SSS format

## Optional Features

Function	Result
utcTimestamp()	Returns The current GMT date and time in the yyyyMMdd-HH:mm:ss format
dateOnly()	Returns The current GMT date in the yyyyMMdd format
monthYear()	Returns the current GMT date in the yyyyMM format
utcTimeOnly()	Returns the current GMT time in the HH:mm:ss format
utcTimeMsOnly()	Returns the current GMT time in the HH:mm:ss.SSS format
utcTZTimeOnly()	Returns the current GMT time with time zone id in the HH:mm:ssZ format
utcTZTimestamp()	Returns the current GMT date and time with time zone id in the yyyyMMdd-HH:mm:ssZ format
utcTZTimestampMs()	Returns the current GMT date and time with time zone id in the yyyyMMdd-HH:mm:ss.SSSZ format
utcTZTimeOnlyMs()	Return the current GMT time with time zone id in the HH:mm:ss.SSSZ format
currentTimeMillis()	Returns the current time in milliseconds
nanoTime()	Returns the current time in nanoseconds
localTimestampMs()	Returns the current Local date and time in the yyyyMMdd-HH:mm:ss.SSS format
localTimestamp()	Returns the current Local date and time in the yyyyMMdd-HH:mm:ss format
localDateOnly()	Returns the current Local date in the yyyyMMdd format
localMonthYear()	Returns the current Local date in the yyyyMM format
localTimeOnly()	Returns the current Local time in the HH:mm:ss format
localTimeMsOnly()	Returns the current Local time in the HH:mm:ss.SSS format
localTZTimeOnly()	Returns the current Local time with time zone id in the HH:mm:ssZ format

## Optional Features

Function	Result
localTZTimestamp()	Returns the current Local date and time with time zone id in the yyyyMMdd-HH:mm:ssZ format
localTZTimestampMs()	Returns the current Local date and time with time zone id in the yyyyMMdd-HH:mm:ss.SSSZ format
localTZTimeOnlyMs()	Return The current Local time with time zone id in the HH:mm:ss.SSSZ format
compareUTCTimeToCurrent(String utcTime)	<p>Compares the specified time to the current time.</p> <p>Returns a negative value if the time is earlier than the current time, zero if the times are the same, and a positive value if the time is later than the current time. For example:</p> <pre>compareUTCTimeToCurrent(myTime) &lt; 0</pre>
compareUTCTimes(String utcTime1, String utcTime2)	<p>Compares the two times.</p> <p>Returns a negative value if the first time is earlier than the second time, zero if the times are the same, and a positive value if the first time is later than the second time. For example:</p> <pre>compareUTCTimes(myTime, '20120316-00:00:01') &gt; 0</pre>

## Unique ID (UUID) Functions

These functions generate UUIDs in various forms. A UUID (Universally Unique Identifier) is represented by 32 hexadecimal digits, displayed in five groups separated by hyphens, in the form 8-4-4-4-12 for a total of 36 characters (32 alphanumeric characters and four hyphens).

An example UUID: 12345678-1234-1234-1234-1234567890AB

Function	Result
uuid	Returns a full 32 digit (36 with the dashes) UUID
uuid_hash	Returns the hash of the whole UUID as an integer in string format
uuid_timestamp	Returns the first three sections of the UUID (16 digits) in reverse order, without dashes

Function	Result
uuid_<sections>	Returns an identifier made up of the specified sections of the original UUID. For example UUID_134 provides the first, third and forth sections of the five section UUID.

## Stored Message Query Functions

To be able to use these message query functions, any required tags should be stored using [ActionStoreTags](#).

Function	Result
boolean relatedRequestExists(String ClOrdID)	Returns true if a message related to the supplied ClOrdID has been previously stored
String getRequestMessageFieldValue(String clOrdID, int tag)	Returns the value of the field with the supplied tag from the message data corresponding to ClOrdID

The following example demonstrates the usage of these functions:

```
<Rule direction="outbound" description="Replace OrdStatus and ExecType in ER">
  <Conditions>
    <Inclusions>
      <ConditionMatchAny tags="35" values="8"/>
      <ConditionMatchAny tags="39" values="6"/>
      <ConditionEvaluate values="relatedRequestExists(ClOrdID) and
                                matches(getRequestMessageFieldValue(ClOrdID, 35), 'G')"/>
    </Inclusions>
    <Exclusions/>
  </Conditions>
  <Actions>
    <ActionSetExpression tag="39" expression="replaceFirst(OrdStatus, '6', 'E')"/>
    <ActionSetExpression tag="150" expression="replaceFirst(ExecType, '6', 'E')"/>
  </Actions>
</Rule>
```

### 6.2.4. Conditions

A rule is made up of a `Conditions` element and an `Actions` element. When the conditions are satisfied then the actions are executed. Conditions consist of a set of `Inclusions` and `Exclusions` elements. For the conditions to be satisfied (and the actions executed), all individual inclusion conditions must be met and no individual exclusion condition must be met.

The following is an example showing how conditions are configured:

```
<Rule id="inbound_validation" description="Price missing on Limit Order">
```



## Optional Features

```
<Conditions>
  <!--Look for Limit Order type messages-->
  <Inclusions>
    <ConditionMatchMsgType values="D,F,G" />
    <ConditionMatchAny tags="40" values="2" />
  </Inclusions>
  <!--Where the price field is missing-->
  <Exclusions>
    <ConditionFieldsPresent tags="44" />
  </Exclusions>
</Conditions>
...
</Rule>
```

For the actions on this rule to execute, the MsgType(35) field must be: D, F or G and the OrdType(40) field must be Limit(2) and the Price(44) must be missing.



### Note

Conditions are not required. If no conditions are specified then the rule's actions will always be executed.

## 6.2.4.1. Built-in Conditions

### ConditionMatchMsgType

Is satisfied when the MsgType(35) field is one of the specified values.

Parameters:

- **values** - a list of message types to match on.

```
<ConditionMatchMsgType values="f,h" />
```

### ConditionFieldsPresent

Is satisfied if a message contains any of the fields listed.

Parameters:

- **tags** - a list of fields to check for.

```
<ConditionFieldsPresent tags="11,55" />
```

## ConditionFieldsUnique

Is satisfied if the values of the specified fields are unique within those that have already been stored. The data store that contains these fields will be cleared out when the EOD process runs.

Parameters:

- **tags** - the fields to check for uniqueness.

```
<RulesPack id="validate-order-fields">
  <Rule description="Reject Orders with Duplicate ClOrdIDs">
    <Conditions>
      <Inclusions>
        <ConditionMatchMsgType values="D"/>
      </Inclusions>
      <Exclusions>
        <ConditionFieldsUnique tags="11" updateStore="true" />
      </Exclusions>
    </Conditions>
    <Actions>
      <ActionReject rejectType="OrderReject" reason="Duplicate ClOrdIDs" />
    </Actions>
  </Rule>
</RulesPack>
```

## ConditionMatchAny

Is satisfied if any of the specified fields have any of the specified values.

Parameters:

- **tags** - the fields to check.
- **values** - the values to check for in the fields.

```
<ConditionMatchAny tags="39" value="2,4"/>
```

## ConditionEvaluate

Is satisfied if the expression evaluates to true. For information on the expression syntax and a list of supported functions, see the [Rules Expression Language](#) section.

Parameters:

- **values** - the expression to evaluate.

```
<!-- Evaluate if 1st instance of tag523 (nested repeating group) equals "BIC12345" -->
<ConditionEvaluate values="matches(fields[453][0][802][0][523], 'BIC12345')"/>
```

## ConditionValidChar

Is satisfied if all of the values of the fields corresponding to `tags` are valid characters and they match one of `values`.

Parameters:

- **tags** - the fields to validate.
- **values** - allowable character values.

```
<ConditionValidChar values="Y,N" tags="43" />
```

## ConditionValidInt

Is satisfied if all of the values of the fields corresponding to `tags` are valid integers and are between the `minValue` and `maxValue` (if supplied).

Parameters:

- **tags** - the fields to validate.
- **minValue** - the minimum allowable value for each of these integer fields.
- **maxValue** - the maximum allowable value for each of these integer fields.

```
<ConditionValidInt minValue="1" maxValue="1000" tags="38" />
```

## ConditionValidUTCTimeStamp

Is satisfied if all of the values of the fields corresponding to `tags` are valid UTC timestamps.

Parameters:

- **tags** - the fields to validate.

```
<ConditionValidUTCTimeStamp tags="52,60" />
```

## ConditionValidFIX

Is satisfied if the message contains valid FIX fields. The default behavior can be overridden by modifying the XML files in *resources/validation* and specifying the path to a modified file in **validationRulesFileName** attribute

Parameters:

## Optional Features

- **validationRulesFileName** - the validation rules file to validate against. The value of this file is a path relative to *resources/validation* directory. If not specified, validation is performed against the dictionary of the corresponding FIX session.
- **validateDataTypesOnly** - if `true`, validates the data types only instead of data types and required fields.

```
<RulesPack id="inbound-validation">
  <Rule direction="inbound" description="Reject Invalid Messages">
    <Conditions>
      <Inclusions/>
      <Exclusions>
        <!--Must be an Exclusion to fire when messages do not pass validation-->
        <ConditionValidFIX />
      </Exclusions>
    </Conditions>
    <Actions>
      <ActionReject rejectType="SessionReject"/>
    </Actions>
  </Rule>
</RulesPack>
```

## ConditionStaleTime

Is satisfied if the limit specified has been exceeded at the time the message is processed.

Parameters:

- **tag** - the tag whose value specifies the time to check against, `SendingTime(52)` for example.
- **staleTimeLimit** - if the amount of time (in milliseconds) between the time found in the message and the current time exceeds this value, then the condition is satisfied.
- **staleIfTagMissing** - if set to `true`, the condition is also satisfied if the tag is missing from the message.

```
<ConditionStaleTime staleIfTagMissing="true" staleTimeLimit="5000" tag="52" />
```

## ConditionTimeInterval

Is satisfied if the time that the message is processed is between the limits of the time interval specified. Can be used for time-based routing or validation.

Parameters:

- **fromDayOfWeek** - the day of the week that the interval starts.
- **fromHour** - the hour of the day that the interval starts.
- **fromMinute** - the minute of the hour that the interval starts.

- **toDayOfWeek** - the day of the week that the interval finishes.
- **toHour** - the hour of the day that the interval finishes.
- **toMinute** - the minute of the hour that the interval finishes.
- **timeZone** - the timezone in which the interval is specified.
- **dayBoundary** - whether the interval crosses over a day boundary.

```
<ConditionTimeInterval dayBoundary="true" timeZone="America/Chicago">  
  <From dayOfWeek="1" hour="8" minute="0" />  
  <To dayOfWeek="5" hour="17" minute="0" />  
</ConditionTimeInterval>
```

## GenericCondition

`GenericCondition` allows custom conditions to be added to the Node. A custom condition class must implement the [ICondition](#) interface. In particular, `ICondition.isSatisfied(IFIXMessage message)` should return `true` if the condition is met and `false` otherwise. Custom properties can be passed to the condition by defining one or more nested `Property` elements.

For an example see *templates/src/com/camerontec/catalys/server/rules/test/SampleCustomCondition.java*

Parameters:

- **class** - the fully qualified class name of the custom condition class.

```
<GenericCondition class="com.camerontec.catalys.server.rules.test.SampleCustomCondition">  
  <Property name="property1" value="value1" />  
</GenericCondition>
```

## 6.2.5. Actions

A rule's actions are executed when its conditions are satisfied. Actions are executed in the order in which they appear under their parent `Actions` element.

Certain actions (reject, route, discard) are considered *final*; after such an action is executed, the message does not pass through any subsequently defined rules. These types of actions should be the last action configured in the `Actions` element.

### 6.2.5.1. Action Definitions

#### ActionAddTags

Adds, replaces or appends a field to a message.

Parameters:

- **tags** - the fields to add, replace or append to the message.
- **value** - the value for the fields.
- **append** - if `true`, the tag is appended to the message, so existing values of the tag are preserved in the message. If `false`, then existing fields are replaced with the value.

```
<ActionAddTags value="compid2" append="true" tags="106" />
```

### ActionRemoveTags

Removes a list of fields from a message.

- **tags** - the fields to remove from a message.

```
<ActionRemoveTags tags="109" />
```

### ActionCopy

Copy the value of one field to another field, and overwrite all instances of the field if it already exists in the message.

Parameters:

- **tagFrom** - the source field to copy the value from.
- **tagsTo** - the target field(s) to copy the value to.

```
<ActionCopy tagFrom="11" tagsTo="58" />
```

### ActionReplaceAll

Performs a regular expression replacement on the value of a field.

Parameters:

- **tags** - the fields to replace.
- **regex** - the regular expression pattern to match on.
- **replacement** - the value to be substituted when the pattern is found.

```
<!--Find 'Order' replace with 'Clordid' in tag58-->
```

## Optional Features

```
<ActionReplaceAll tags="58" regex="Order" replacement="Clordid"/>
<!--Remove any spaces in tag51 and tag100-->
<ActionReplaceAll tags="1, 100" regex="\s" replacement=""/>
<!--Use pattern group matching and $ characters to the swap order of
text before and after period in tag48 -->
<ActionReplaceAll tags="48" regex="(.)\\.(.*)" replacement="$2.$1"/>
```

### ActionDiscard

Silently discards the message. No reject message is sent.

Parameters: none.

```
<ActionDiscard />
```

### ActionMapper

Replaces the value of a field using a lookup properties table. If a value is not found in the lookup table, the field is left unchanged. The mapping table can be applied in either direction.

Parameters:

- **tags** - the fields to perform the lookup on.
- **mappingTableRefId** - the mapping table ID to use.
- **leftToRight** - should the mapping be from left to right, or right to left. Defaults to `true`.

In this example, `ActionMapper` is used to map security identifiers. Outbound messages will have the SEDOL format replaced with with ISIN format and the reverse mapping applied to inbound messages. Note that the same mapping table is used by both `ActionMapper` actions.

```
<Application id="FIX_ROUTER">
  <Properties id="sedol-to-isin">
    <Property name="2543495" value="US11041RAL24" />
    <Property name="B0V3Xd1" value="AU000000GFF8" />
  </Properties>
  ...
  <RulesPack id="securityid-mapping">
    <Rule direction="outbound" description="SEDOL to ISIN">
      <Conditions>
        <Inclusions/>
        <Exclusions/>
      </Conditions>
      <Actions>
        <ActionMapper mappingTableRefId="sedol-to-isin" tags="SecurityID" leftToRight="true"/>
      </Actions>
    </Rule>
    <Rule direction="inbound" description="ISIN to SEDOL">
      <Conditions>
```

```

        <Inclusions/>
        <Exclusions/>
    </Conditions>
    <Actions>
        <ActionMapper mappingTableRefId="sedol-to-isin" tags="SecurityID" leftToRight="false"/>
    </Actions>
</Rule>
</RulesPack>
...
</Application>

```

## ActionFileLookup

This action performs a lookup via a [FileLookupService](#) element and writes the result to one or more message fields. If the message doesn't contain the necessary lookup values, or if the lookup value is not found in the service's data, the message is left unchanged or can optionally be rejected.

Parameters:

- **fileLookupServiceId** - the `FileLookupService` instance to use.
- **lookupKey** - the column(s) in the lookup data that contain the lookup value.
- **lookupTags** - a comma-separated list of message field(s) that contain the lookup value.
- **resultIndex** - the column index containing the value to store in the field(s).
- **tags** - a comma-separated list of field(s) to replace with data returned by the lookup.
- **rejectWhenNoResult** - if `true`, the message will be rejected if the lookup returns no results. Defaults to `false`.

In this example, the data from the service named "symbolConversion" is used to convert symbols. The first column of the data (index 0) is the used as the lookup key. `Symbol(55)` is used as the lookup value. The second column (index 1) of the data's value is returned. The values of fields `Symbol(55)` and `custom(6055)` are replaced with the data from the second column.

```

<ActionFileLookup
  fileLookupServiceId="symbolConversion"
  lookupKey="0"
  lookupTags="55"
  resultIndex="1"
  tags="55,6055" />

```

See the documentation for [FileLookupService](#) for more detailed examples.

## ActionSetExpression

Sets the value of a field to the result of a [Rules Engine Expression](#).



Parameters:

- **tag** - the field to set the value of.
- **expression** - the [expression](#) to set the value to.

In this example, `ActionSetExpression` is used to calculate a 1% commission value on all inbound fills by multiplying the `(Price(44) * LastQty(32) * .01)` and setting it to `Commission(12)`.

```
<RulesPack id="RulesInbound">
  <Rule direction="inbound" description="Set Commission on Fills">
    <Conditions>
      <Inclusions>
        <ConditionMatchMsgType values="8" />
        <ConditionMatchAny tags="150" values="1,2" />
      </Inclusions>
      <Exclusions />
    </Conditions>
    <Actions>
      <ActionSetExpression tag="12" expression="tag44*tag32*.01"/>
    </Actions>
  </Rule>
</RulesPack>
```

## ActionTimezoneConversion

Converts the specified date/time fields from one timezone to another.

Parameters:

- **tags** - the time fields to convert.
- **fromTimezone** - Abbreviation of the timezone being converted from. GMT is assumed if not specified.
- **toTimezone** - Abbreviation of the timezone being converted to. GMT is assumed if not specified.
- **validateFormat** - If set to `true` all specified timezone values will be validated. if the format is invalid, the execution of this action and all further actions for this rule will not be executed.
- **dateFormat** - The default format for the CRE is: `yyyyMMdd-HH:mm:ss`. This can be overridden using this attribute. However, values of all specified tags must adhere to this format. If at least one value's format does not match, an exception is thrown and the action has no effect.

```
<!-- See CatalysConfig.dtd for a list of valid timezones. -->
<ActionTimezoneConversion
  tags="122,52,60"
  validateFormat="false"
  toTimezone="America/Eastern"
  fromTimezone="GMT" />
```

## ActionStripChars

Strips the characters in the specified range from a value of given field.

Parameters:

- **tags** - the list of fields to strip characters from.
- **beginIndex** - character index at which the stripping of chars begins. If the value of this attribute is negative, 0 is assumed.
- **endIndex** - index at which stripped chars end. If the value of this attribute is negative or it is lower than `beginIndex`, the end of the processed tag's value is assumed.

```
<ActionStripChars beginIndex="2" endIndex="4" tags="106" />
```

## Store and Restore Actions

These actions allow certain message fields to be stored and indexed, so that they can be retrieved and restored to enhance future related messages. Fields that have previously been stored can be removed from the store based on certain conditions (an order being cancelled for example).

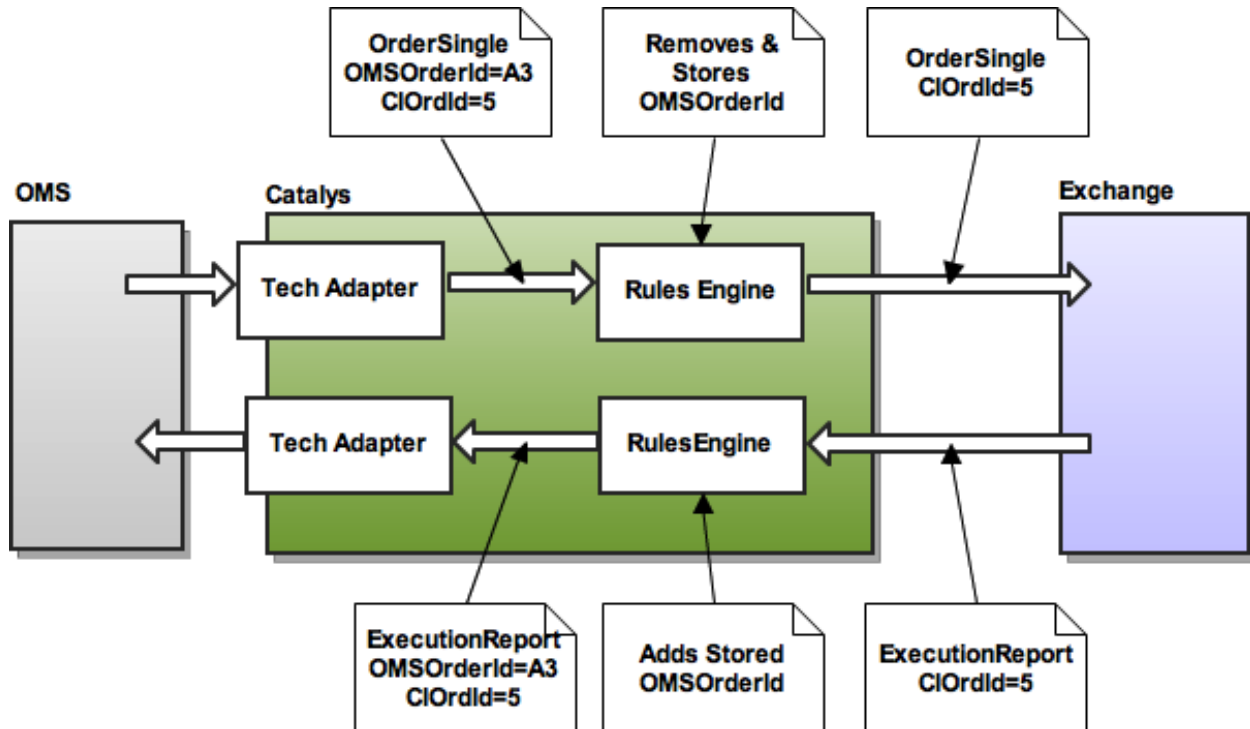
Store and Restore supports the following types of messaging:

- Orders and Execution Reports - keyed off `ClOrdID`
- Quote Request and Response - keyed off `QuoteReqID` or `QuoteID`
- Allocation Instruction - keyed off `AllocID`
- Indication of Interest (IOIs) - keyed off `IOIID`
- Reject Messages - keyed off `MsgSeqNum`

Each `CatalysRulesEngine` element has its own message store, so to enable storing of outbound messages, and restoring into inbound messages, a single `CatalysRulesEngine` must be used. The [MessageProcessorReference](#) element can be used to achieve this.

The diagram below shows the use of Store and Restore where an OMS uses a custom field, `OMSOrderID`, to track orders and execution reports, but the counterparty does not accept that field. On outbound orders, the `ActionStoreTags` and `ActionRemoveTags` actions can store the value of the field, then remove it from the message. When receiving an execution report, `ActionAddStoredTags` can be used to look up the stored value by `ClOrdID` and add it to the execution report before it is processed by the OMS.

## Optional Features



The result of this configuration is that all messages passing to and from the OMS can contain the OMSOrderId tag, while messages to and from the counterparty do not.

## ActionStoreTags

Stores fields from a message for later retrieval by `ActionAddStoredTags`.

If the `persistPath` attribute is set on the [CatalysRulesEngine](#) element, then the messages stored will be persisted between restarts of the application, and replicated between Nodes in a High Availability cluster.

In this case stored messages should be cleared (so that they do not grow indefinitely), by an [end-of-day](#) (EOD) task or command, which removes closed orders from the CRE store; or by using the [ActionUnstoreTags](#) action.

Parameters:

- **tags** - a list of fields to store. Tags used as indexes are automatically stored. Required.
- **indexByMsgSeqNum** - if set to `true`, messages are also indexed by the `MsgSeqNum` tag, so that when `Reject` or `BusinessMessageReject` messages are received, they can be enriched with the information from the original message. Defaults to `false`.

## ActionAddStoredTags

Adds fields to the message, if they have been previously stored by `ActionStoreTags`.

If the value of a field in the original message is modified by a subsequent related message, then it is the modified value which is added to the message by this action. If the field is not present in any subsequent related messages, then its original value is added. For example, if an order contains a custom field and the value of this custom field has a different value in a subsequent amendment to the order, then this subsequent value is used.

Parameters:

- **tags** - the fields that identify the fields to copy from the store into the message.

```
<RulesPack id="store-and-restore-symbol-tag">
  <Rule direction="outbound" description="Store Order Symbol">
    <Conditions>
      <Inclusions>
        <ConditionMatchMsgType values="D"/>
      </Inclusions>
      <Exclusions/>
    </Conditions>
    <Actions>
      <!-- store the stock symbol -->
      <ActionStoreTags tags="Symbol"/>
    </Actions>
  </Rule>
  <Rule direction="inbound" description="Stored symbol update in ExecutionReport">
    <Conditions>
      <Inclusions>
        <ConditionMatchMsgType values="8"/>
      </Inclusions>
      <Exclusions/>
    </Conditions>
    <Actions>
      <!-- add the stored stock symbol into the execution report -->
      <ActionAddStoredTags tags="Symbol"/>
    </Actions>
  </Rule>
</RulesPack>
```

## ActionUnstoreTags

Removes records that have been previously stored via the `ActionStoreTags` action. For example, this action can be used to remove closed orders from the store:

Parameters:

- **tags** - a list of fields, whose values uniquely identify the records in the store to delete. For example: ClOrdID for orders, QuoteID for quotes.

```
<!-- unstore closed orders -->
<Rule direction="inbound">
  <Conditions>
    <Inclusions>
      <ConditionMatchMsgType values="8"/>
```

```
<ConditionMatchAny tags="39" value="2,4" />
</Inclusions>
<Exclusions/>
</Conditions>
<Actions>
  <ActionUnstoreTags tags="11" />
</Actions>
</Rule>
```

### ActionDropCopyTo

Creates a copy of a message and sends it to another session.

See the [Drop Copy](#) documentation for more information on this.

Parameters:

- **destination** - the session to send the copy of the message to.

This should be in the form <external party>~<internal party>. For example, for a buy-side session this might be "broker~fund". If the parties of the session have sub IDs and location IDs, then these should be separated by a dot, i.e. Comp.Sub.Loc. For example: "broker.departmentA.london~fund.level2.paris".

The message will pass through any source message processors that are configured on the destination session.

The following example demonstrates sending a drop copy of all fills from a broker's session with a client to their compliance department:

```
<Config>
  <Application id="FIX_ROUTER">
    <RulesPack id="broker_rules">
      <Rule direction="inbound" description="Drop Copy Fills to Compliance">
        <Conditions>
          <Inclusions>
            <ConditionMatchMsgType values="8" />
            <ConditionMatchAny tags="150" values="1,2" />
          </Inclusions>
          <Exclusions/>
        </Conditions>
        <Actions>
          <ActionDropCopyTo destination="COMPLIANCE.BANK.PARIS~BROKER.EXCHANGE.LONDON" />
        </Actions>
      </Rule>
    </RulesPack>
  </Application>
  <Sessions>
    <Session counterpartycompid="CLIENT" compid="BROKER" fixversion="4.4">
      <Connections>
        <SocketConnection id="scl" hostname="localhost" port="4000"/>
      </Connections>
      <SessionManager>
        <SourceMessageProcessors/>
      </SessionManager>
    </Session>
  </Sessions>
</Config>
```

## Optional Features

```
<ListenerMessageProcessors>
  <CatalysRulesEngine id="cre" rulesPackId="broker_rules"/>
</ListenerMessageProcessors>
</SessionManager>
</Session>
<Session counterpartycompid="COMPLIANCE"
  counterpartysubid="BANK"
  counterpartylocationid="PARIS"
  compid="BROKER"
  subid="EXCHANGE"
  locationid="LONDON"
  fixversion="4.4">
  <Connections>
    <SocketConnection id="sc2" hostname="localhost" port="4001"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors/>
    <ListenerMessageProcessors/>
  </SessionManager>
</Session>
</Sessions>
</Application>
</Config>
```

### ActionRouteTo

Routes an inbound message to another session, and can route responses (e.g. an execution report) back to the originating session.

A [Link](#) element must also be configured, linking the sessions. Only inbound messages can be routed using this action.

If more than one connection point is provided, then each one will be tried in order. Once a message has been routed via one of these connection points, all related (cancels, amends, etc.) messages will be routed via the same connection point.

Parameters:

- **selectedConnectionPoint** - A comma-separated list of connection points to route messages to. These should be in the form [external party]~[internal party]. At least one connection point must be provided.
- **autoRoutingResponse** - If set to `true`, messages are routed to the destination session (determined by `selectedConnectionPoint`) and replies are returned back to the source session. Note, that in order to store the routes after restart, you need to set 'directoryName' attribute for [Link](#). Default is `false`.
- **autoRouteSubsequentRequests** - If set to `true` and there is only one `selectedConnectionPoint`, related messages will be automatically routed to the same `selectedConnectionPoint`, even if they are excluded by the configured conditions.

```
<Config>
  <Application id="broker">
    <RulesPack id="RouteToExch1">
```

## Optional Features

```
<Rule direction="inbound" description="Route To exchange1">
  <Conditions>
    <Inclusions />
    <Exclusions />
  </Conditions>
  <Actions>
    <ActionRouteTo selectedConnectionPoint="exchange1~broker" />
  </Actions>
</Rule>
</RulesPack>
<RulesPack id="RouteToExch2">
  <Rule direction="inbound" description="Route To exchange2">
    <Conditions>
      <Inclusions />
      <Exclusions />
    </Conditions>
    <Actions>
      <ActionRouteTo selectedConnectionPoint="exchange2~broker" />
    </Actions>
  </Rule>
</RulesPack>
<Link id="routingTable">
  <SessionGroup>
    <SessionReference id="sessionref1" counterpartycompid="buy2" compid="broker" />
    <SessionReference id="sessionref2" counterpartycompid="buy1" compid="broker" />
  </SessionGroup>
  <SessionGroup>
    <SessionReference id="sessionref3" counterpartycompid="exchange3" compid="broker" />
    <SessionReference id="sessionref4" counterpartycompid="exchange2" compid="broker" />
    <SessionReference id="sessionref5" counterpartycompid="exchange1" compid="broker" />
  </SessionGroup>
</Link>
<Sessions>
  <Session counterpartycompid="buy1" compid="broker" side="buy" fixversion="4.2">
    <Connections>
      <SocketConnection id="conn.buy1" listenport="2000" />
    </Connections>
    <SessionManager>
      <SourceMessageProcessors>
        <CatalysRulesEngine id="CRE1" rulesPackId="RouteToExch2" />
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <MessageProcessorReference id="sharedCRE1" refid="CRE1" />
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
  <Session counterpartycompid="buy2" compid="broker" side="buy" fixversion="4.2">
    <Connections>
      <SocketConnection id="conn.buy2" listenport="2001" />
    </Connections>
    <SessionManager>
      <SourceMessageProcessors>
        <CatalysRulesEngine id="CRE2" rulesPackId="RouteToExch1" />
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <MessageProcessorReference id="sharedCRE2" refid="CRE2" />
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
</Sessions>
```

## Optional Features

```
</SessionManager>
</Session>
<Session counterpartycompid="exchange1" compid="broker" side="sell" fixversion="4.2">
  <Connections>
    <SocketConnection id="conn.exchange1" hostname="localhost" port="2003" />
  </Connections>
  <SessionManager>
    <SourceMessageProcessors />
    <ListenerMessageProcessors />
  </SessionManager>
</Session>
<Session counterpartycompid="exchange2" compid="broker" side="sell" fixversion="4.2">
  <Connections>
    <SocketConnection id="conn.exchange2" hostname="localhost" port="2004" />
  </Connections>
  <SessionManager>
    <SourceMessageProcessors />
    <ListenerMessageProcessors />
  </SessionManager>
</Session>
<Session counterpartycompid="exchange3" compid="broker" side="sell" fixversion="4.2">
  <Connections>
    <SocketConnection id="conn.exchange3" hostname="localhost" port="2005" />
  </Connections>
  <SessionManager>
    <SourceMessageProcessors />
    <ListenerMessageProcessors />
  </SessionManager>
</Session>
</Sessions>
</Application>
</Config>
```

## ActionRouteToAdapter

Routes a message to an adapter.

Parameters:

- **adapterRefId** - the ID of the adapter to route the message to.
- **autoRouteSubsequentRequests** - If set to `true` and there is only one `adapterRefId` listed, related messages will be automatically routed to the same `adapterRefId`, even if they are excluded by the configured conditions.

```
<!-- Route to Tech Adapter based on text in ClordID -->
<RulesPack id="Queue_Selector">
  <Rule direction="inbound" description="Route to Staging MQn">
    <Conditions>
      <Inclusions>
        <ConditionEvaluate values="contains(tag11,'NYDSTHOM1')"/>
      </Inclusions>
      <Exclusions />
    </Conditions>
```



## Optional Features

```
<Actions>
  <ActionRouteToAdapter autoRouteSubsequentRequests="false" adapterRefId="msgOutNYDSTHOM1" />
</Actions>
</Rule>
<Rule direction="inbound" description="Route to Staging MQ">
  <Conditions>
    <Inclusions>
      <ConditionEvaluate values="contains(tag11,'SHNYIIS01A')"/>
    </Inclusions>
    <Exclusions />
  </Conditions>
  <Actions>
    <ActionRouteToAdapter autoRouteSubsequentRequests="false" adapterRefId="msgOutSHNYIIS01A" />
  </Actions>
</Rule>
<Rule direction="inbound" description="defaultQueue" defaultCase="true">
  <Conditions>
    <Inclusions />
    <Exclusions />
  </Conditions>
  <Actions>
    <ActionRouteToAdapter autoRouteSubsequentRequests="false" adapterRefId="msgOut" />
  </Actions>
</Rule>
</RulesPack>
```

### ActionReject

This action sends an outbound reject message and no more actions are processed on the current message, which is then discarded.

`ActionReject` can be used in Exception rules. For example, this can be used to catch an exception thrown by a downstream rule (e.g. lookup fails) or message processor (e.g. target session down) and turn it into a reject. The default behaviour when not using an `ActionReject` is that the exception eventually gets caught by the source session manager and gets turned into a Business Message Reject.

When the `rejectType` is set to `ApplicationReject`, the action dynamically determines the `msgType` of the reject message based on the source message. For example, a new order gets rejected with an execution report, an amend gets rejected with a cancel reject, etc. When this `rejectType` is used, the `reasonCode` configuration attribute is ignored, and a default reject reason value is set in the reject message.

Regardless of whether `ActionReject` is used in an inbound rule, an outbound rule, or an exception rule, the resulting reject message is injected into the source message processor of the originating session, which means that this reject message can be post-processed by outbound rules attached to that session.

A typical use case is to further enrich the reject message with custom or lookup fields.

Parameters:

- **rejectType** - Specifies which type of reject message to send. The following values are accepted: `BusinessReject`, `CancelReject`, `DontKnowTrade`, `MarketDataRequestReject`,

## Optional Features

OrderReject, OrderStatusRequestReject, QuoteReject, QuoteRequestReject, SessionReject, ApplicationReject

- **reason** - The reason the message was rejected.
- **reasonCode** - The code of the reason why the message was rejected.

```
<RulesPack id="validate-order-fields">
  <Rule description="Reject Orders with Duplicate ClOrdIDs">
    <Conditions>
      <Inclusions>
        <ConditionMatchMsgType values="D"/>
      </Inclusions>
      <Exclusions>
        <ConditionFieldsUnique tags="11" updateStore="true" />
      </Exclusions>
    </Conditions>
    <Actions>
      <ActionReject rejectType="OrderReject" reason="Duplicate ClOrdIDs" />
    </Actions>
  </Rule>
</RulesPack>
```

## ActionRulesPack

This action allows the conditional execution of other rules packs, and gives the ability to reuse rules. When this action is executed, the rules in the referenced rules pack are run, followed by any remaining rules in the original rules pack.

Parameters:

- **refid** - the ID of the rules pack to run.

The following example shows how a global rules pack could be reused by a different rules pack. The "GLOBAL\_RULES" rules pack is executed first, and then the remaining rules in the "SESSION\_ABC" pack are executed.

```
<RulesPack id="SESSION_ABC">
  <!--Call to Global Rules first-->
  <Rule direction="inbound" description="Call Global Rules">
    <Conditions>
      <Inclusions/>
      <Exclusions/>
    </Conditions>
    <Actions>
      <ActionRulesPack refid="GLOBAL_RULES"/>
    </Actions>
  </Rule>
  <!--Rules specific to Session ABC-->
  <Rule direction="inbound" description="SESSION_ABC Enrichment">
    <Conditions>
```

```

    <Inclusions/>
    <Exclusions/>
  </Conditions>
  <Actions>
    <ActionAddTags tags="100" values="DEST_1" />
  </Actions>
</Rule>
</RulesPack>
<RulesPack id="GLOBAL_RULES">
  <Rule>
    <Conditions>
      <Inclusions/>
      <Exclusions/>
    </Conditions>
    <Actions>
      <ActionAddTags tags="1" values="FIRM_A" />
    </Actions>
  </Rule>
</RulesPack>

```

## ActionLog

This action logs a message at a specified log level and can include the FIX message that triggered it.

Parameters:

- **logLevel** - Specifies at which level to log the message at. The following values are accepted: `ERROR`, `WARN`, `INFO`, `DEBUG`, `TRACE`. Defaults to `INFO`.
- **message** - The message to log.
- **includeFIXMsg** (Optional) - Should the FIX message that triggered the action be included in log message. Defaults to `false`.

```
<ActionLog logLevel="INFO" message="Message being routed :" includeFIXMsg="true" />
```

## GenericAction

`GenericAction` allows custom actions to be added to the Node. A custom action class must implement the [IAction](#) interface. In particular, `IAction.apply(IFIXMessage message)` should execute the steps of the custom action. [ActionBase](#) is a useful base class for custom actions. A `GenericAction` should be considered as an alternative to using `ActionSetExpression` with complex expressions that do not provide adequate performance.

For an example see *templates/src/com/camerontec/catalys/server/rules/test/SampleCustomAction.java*

Parameters:

- **class** - the fully qualified class name of the custom action class.

```
<GenericAction
  id="customAction"
  class="com.camerontec.catalys.server.rules.test.SampleCustomAction"/>
```

## 6.2.6. Message Validation

The rules engine comes with built-in support to validate messages against a specific FIX version. This functionality is available by using the `ConditionValidFIX` element. This is especially useful in a test or QA environment to flag when invalid messages are created. Validation requires inspecting the contents of every field in a message which increases per message latency.

The following checks are performed during FIX version validation:

- Required fields for the message type are present (if the `validateDataTypesOnly` attribute isn't set to `false`).
- The value of each field is a valid instance of the data type for that field.
- The value is within the range, or set of values specified by the FIX specification.

## Configuration Example

To validate inbound messages, add a rule using `ConditionValidFIX` and `ActionReject`.

```
<RulesPack id="inbound-validation">
  <Rule direction="inbound" description="Reject Invalid Messages">
    <Conditions>
      <Inclusions/>
      <Exclusions>
        <!--Must be an Exclusion to fire when messages do not pass validation-->
        <ConditionValidFIX />
      </Exclusions>
    </Conditions>
    <Actions>
      <ActionReject rejectType="SessionReject"/>
    </Actions>
  </Rule>
</RulesPack>
```

To validate outbound messages, configure the set the `validateDataTypesOnly` attribute to `true`, since outbound messages will not have required fields such as `BeginString` when they are processed by the CRE.

By default, `ConditionValidFIX` validates against the FIX version of the current session, using the dictionary files. You can override this default behaviour by specifying the path to an alternative validation rules file located in the `resources/validation` directory in the `validationRulesFileName` attribute.

## 6.3. File Lookup Service

### 6.3.1. Introduction

The `FileLookupService` provides a centralized interface to data in a delimited text file. This data can be used by [ActionFileLookup](#) in the [Catalys Rules Engine](#) and by a custom message processor. The service parses data from user-defined columns in the text file into key-value pairs — the caller provides the key and the service returns the corresponding value.

### 6.3.2. Basic Configuration Example

In this scenario, we are converting symbols in inbound messages prior to routing to another session. Our sample data file:

```
IBM,IBM.X
MSFT,MSFT.X
ORCL,ORCL.X
```

To configure the service, first determine which column(s) will serve as the *lookup key*. In this example, the first column (index 0) contains the symbol we are converting from, so this is our lookup key. Now we must choose the *result* column — this column holds the values we want to retrieve, which is the second column (index 1). Our configuration looks like this:

```
<Config>
  <Application id="app">
    <Services>
      <FileLookupService
        id="symbolConversion"
        fileName="/tmp/symbols.csv"
        delimiter=","
        lookupKeys="0"
        resultIndexes="1" />
    </Services>
  </Application>
</Config>
```

For a complete list of configuration attributes, see the [FileLookupService Configuration Reference](#).

When the service starts, it will parse each line of the file and load all key-value pairs into memory (only columns that are configured are loaded; other columns are ignored). The next step is to configure the Catalys Rules Engine to actually use the data. This is done via the `ActionFileLookup`, which we'll configure to take the value of tag 55 from an incoming message, lookup this value in the `FileLookupService` and overwrite tag 55 with the result:

```
<RulesPack id="outboundRules">
```

```

<Rule direction="outbound" description="Symbol conversion">
  <Conditions>
    <Inclusions />
    <Exclusions />
  </Conditions>
  <Actions>
    <ActionFileLookup fileLookupServiceId="symbolConversion"
                      lookupKey="0"
                      lookupTags="55"
                      resultIndex="1"
                      tags="55"/>
  </Actions>
</Rule>
</RulesPack>

```

The values specified in `lookupKey` and `resultIndex` must be one of the the `lookupKeys` and `resultIndexes` configured in the corresponding service.

### 6.3.3. Configuring Multiple Keys and Results

A `FileLookupService` can be configured with multiple lookup keys and result columns. This is useful when you have a single data file containing different data mappings — one service can be configured to accommodate multiple `ActionFileLookups` that refer to different columns in the data file. Furthermore, one lookup key can be comprised of multiple columns. This is referred to as a *composite key* and is notated as  $\{x, y, \dots, n\}$  where  $x$  and  $y$  are column indexes. A composite key is used when the values of multiple tags are used to lookup a single result value.

To illustrate these concepts, we'll convert symbols based on two criteria: the order type and the destination party. Here is the CSV file:

```

IBM,1,IBM.W,IBM.X
IBM,2,IBM.Y,IBM.Z
MSFT,1,MSFT.W,MSFT.X
MSFT,2,MSFT.Y,MSFT.Z

```

For one counterparty, we must convert to the symbol in column 2; for another party, column 3:

```

<Config>
  <Application id="app">
    <Services>
      <FileLookupService
        id="symbolConversion"
        fileName="/tmp/symbols.csv"
        delimiter=","
        lookupKeys="{0,1}"
        resultIndexes="1,2"
      />
    </Services>
  </Application>
  <RulesPack id="outboundRulesForParty1">

```

```

<Rule direction="outbound" description="Symbol conversion for party1">
  <Conditions>
    <Inclusions />
    <Exclusions />
  </Conditions>
  <Actions>
    <ActionFileLookup fileLookupServiceId="symbolConversion"
                      lookupKey="{0,1}"
                      lookupTags="55,40"
                      resultIndex="1"
                      tags="55" />
  </Actions>
</Rule>
</RulesPack>
<RulesPack id="outboundRulesForParty2">
  <Rule direction="outbound" description="Symbol conversion for party2">
    <Conditions>
      <Inclusions />
      <Exclusions />
    </Conditions>
    <Actions>
      <ActionFileLookup fileLookupServiceId="symbolConversion"
                        lookupKey="{0,1}"
                        lookupTags="55,40"
                        resultIndex="2"
                        tags="55" />
    </Actions>
  </Rule>
</RulesPack>
</Application>
</Config>

```

The `lookupKey` for each action is composite — two pieces of data are required in order to perform the lookup. Therefore we specify two `lookupTags`: `Symbol` and `OrdType`. Note that these must be listed in the same order as the `lookupKey` columns, and columns in a composite `lookupKey` should always be specified in ascending order. This configuration implies that tag 55 corresponds to column 0 and tag 40 corresponds to column 1.

### 6.3.4. Lookup from a Custom Message Processor

An `ActionFileLookup` is not the only component that can make use of a `FileLookupService`. A custom message processor can access the service via the [ServiceRegistry](#):

```

import com.camerontec.catalys.core.message.FIXMessage;
import com.camerontec.catalys.server.lookup.FileLookupKey;
import com.camerontec.catalys.server.lookup.FileLookupService;
import com.camerontec.catalys.server.processor.FIXListenerBase;
import com.camerontec.catalys.util.data.MissingDataException;
import com.camerontec.catalys.util.messaging.MessageEvent;
import com.camerontec.catalys.util.service.ServiceRegistry;
import gnu.trove.TIntIntHashMap;

public class LookupProcessor extends FIXListenerBase

```

```

{
    private final FileLookupKey lookupKey;
    private final TIntIntHashMap lookupKeyToTag;
    private FileLookupService service;

    public LookupProcessor()
    {
        // lookup using columns 0 and 1
        lookupKey = new FileLookupKey(0, 1);

        // map column 0 -> Symbol and column 1 -> OrdType
        lookupKeyToTag = new TIntIntHashMap();
        lookupKeyToTag.put(0, 55);
        lookupKeyToTag.put(1, 40);
    }

    @Override
    public boolean openComponent() throws Exception
    {
        service = (FileLookupService)ServiceRegistry.getInstance()
            .getService("symbolConversion");
        return super.openComponent();
    }

    @Override
    public void onMessageFromFix(MessageEvent event, IFIXMessage message)
        throws MissingDataException
    {
        if (message.hasField(55))
        {
            // retrieve the value in column 2
            final String newSymbol = service.lookup(lookupKey, lookupKeyToTag, 2, message);

            if (newSymbol != null)
            {
                message.setValue(55, newSymbol);
            }
        }
    }
}

```

### 6.3.5. Labels

To simplify the configuration of `FileLookupService` and `ActionFileLookup` via the Dashboard, column labels can be defined for the `lookupKeys` and `resultIndexes` values. This allows a Dashboard user to refer to these columns as names rather than numeric column indexes. For example:

```

IBM,1111,2222
MSFT,3333,4444
ORCL,5555,6666

```

```
<FileLookupService
```



```

id="symbolConversion"
fileName="/tmp/symbols.csv"
delimiter=","
lookupKeys="0,1"
lookupKeyLabels="symbol,isin"
resultIndexes="2"
resultIndexLabels="sedol" />

```

With this configuration, Dashboard users will see the available `lookupKeys` as `symbol` and `isin`, and the available `resultIndexes` will be seen as `sedol`.



### Note

The order of the labels must match the order of the specified column indexes.

## 6.4. Scheduled Order Cache

### 6.4.1. Introduction

The Scheduled Order Cache is a built-in message processor that provides functionality to cache order messages on a session that is not active, and then release the messages at a certain time in the future. There is support for replacing and cancelling orders that are held in the cache.

### 6.4.2. Configuration

The `ScheduledOrderCacheProcessor` can only be configured on the source (outbound) side of a session's `SessionManager` element. It will throw a configuration exception if configured as a listener. Also, only one `ScheduledOrderCacheProcessor` instance may be configured per session. No message processor references are allowed.

The following shows a `ScheduledOrderCacheProcessor` element configured on a session with a `Properties` section and a nested `CatalysRulesEngine`. This instance will cache messages where `tag100='EXCH1'` between 15:15 and 08:00 (next day) and all other messages between 15:30 and 08:30 (next day).

```

<Session counterpartycompid="ROUTING_HUB" compid="BROKER_ABC" fixversion="4.2">
  ...
  <SessionManager>
    <SourceMessageProcessors>
      <ScheduledOrderCacheProcessor id="soc1"
                                   dataDirectory="./data/ordercache/EXCHANGE_A"
                                   msgTypes="D,F,G">

```

## Optional Features

```
<CatalysRulesEngine id="cre1" rulesPackId="exchange-rules" />
<Properties id="soc1-props">
  <!--Schedule for EXCH1 messages-->
  <Property name="EXCH1.condition" value="tag100=='EXCH1'" />
  <Property name="EXCH1.schedule" value="08:00,15:15" />
  <!--Default schedule for everything else-->
  <Property name="schedule" value="08:30,15:30" />
</Properties>
</ScheduledOrderCacheProcessor>
</SourceMessageProcessors>
<ListenerMessageProcessors>
</ListenerMessageProcessors>
</SessionManager>
</Session>
```

For a complete list of configuration attributes, see the [ScheduledOrderCacheProcessor Configuration Reference](#).

## Schedule and Condition Configuration Properties

The caching and releasing of messages is controlled by one or more `schedule` and `condition` property elements. The most basic configuration is a single default `schedule` property:

```
<Properties id="cachel-props">
  <!--do not cache msgs between 08:40 to 15:45-->
  <Property name="schedule" value="08:40,15:45" />
</Properties>
```

It is also possible to define multiple conditions and schedules to cache and release different messages at different times. This is accomplished by using `<KEY>.condition` and `<KEY>.condition` property elements.

```
<Properties id="cachel-props">
  <!--EXCH1 Market Order schedule-->
  <Property name="EXCH1_1.condition" value="tag100=='EXCH1' && tag40==1" />
  <Property name="EXCH1_1.schedule" value="07:05,16:15" />
  <!--EXCH2 Non Market Order schedule-->
  <Property name="EXCH1_2.condition" value="tag100=='EXCH1'" />
  <Property name="EXCH1_2.schedule" value="07:30,16:15" />
  <!--default schedule-->
  <Property name="schedule" value="08:00,16:00" />
</Properties>
```

Refer to the [Rules Engine Expression Language](#) section for the information on what is supported in the condition syntax.



### Important

The value of the `schedule` element defines when to NOT cache messages, i.e. when the venue is accepting orders.

## 6.4.3. Behavior of Scheduled Order Cache

NewOrderSingle (D) and OrderCancelReplaceRequest (G) will be cached without acknowledgement when the destination is out of schedule. The messages will be released when the schedule opens.

OrderCancelRequest (F) messages will be handled in two different ways:

- If the order exists in the cache, the entire order chain will be removed and acknowledged via ExecutionReport (8) back to the sender.
- If the order does not exist in the cache, an OrderCancelReject (9) will be sent back to the sender.

Application messages generated by the `ScheduledOrderCacheProcessor` are FIX version compliant for FIX 4.0 to FIX 5.0SP2.



### Note

The implicit assumption is that order cancel replace requests contain the same schedule matching conditions as the originating order. It is recommended that validation be configured to guarantee this is the case.

## 6.5. FIXML Transformer

### 6.5.1. Introduction

The FIXML Transformer is designed to work with certain Catalys Node Technology Adapters. This transformer enables the Node to accept FIXML messages over a technology adapter, transform them into FIX messages which are then sent over a session. It can also transform inbound FIX messages into FIXML and pass them back over a technology adapter or simply log them to a file.

This transformer is designed to work with all FIXML versions and contains the FIXML 4.4 and 5.0 schemas, which may also be freely downloaded from the [FIX Protocol website](#).

## 6.5.2. Configuration

The FIXML transformer can be configured to work with certain technology adapters via the `transformer` attribute by setting it to `com.camerontec.catalys.core.fixml.FIXMLTransformer`.

The following example shows the FIXML transformer configured on the outbound side of a JMS Adapter.

```
<Session compid="CLIENT" counterpartycompid="SERVER" fixversion="4.2">
  <Connections>
    <SocketConnection id="conn" hostname="10.1.1.1" port="2000" />
  </Connections>
  <SessionManager>
    <SourceMessageProcessors>
      <JmsConsumer type="queue" destination="OUTBOUND" factory="primaryQCF" id="idN65562"
        transformer="com.camerontec.util.fixml.FIXMLTransformer">
        <Properties>
          <Property name="ignoreUndefinedElements" value="true"/>
          <Property name="ignoreUndefinedAttributes" value="true"/>
          <Property name="ignoreUndefinedFIXTags" value="true"/>
        </Properties>
      </JmsConsumer>
    </SourceMessageProcessors>
    <ListenerMessageProcessors/>
  </SessionManager>
</Session>
```

The transformer attribute is parametrized by a nested `Properties` element. The attributes allowed within this nested `Properties` element are:

Attribute	Notes
<code>ignoreUndefinedElements</code>	<p>If enabled, the transformer will ignore FIXML elements (XML tags) that do not appear in the schema when attempting to transform FIXML messages into FIX. If this attribute is not set then the Node will throw an exception when an element that does not appear in the XSD is found in a FIXML message.</p> <p>Optional. Defaults to <code>false</code>.</p>
<code>ignoreUndefinedAttributes</code>	<p>If enabled, the transformer will ignore attributes that do not appear in the schema when attempting to transform FIXML messages into FIX. If this attribute is not set then the Node will throw an exception when an attribute that does not appear in the XSD is found in a FIXML message.</p> <p>Optional. Defaults to <code>false</code>.</p>

Attribute	Notes
ignoreUndefinedFIXTags	<p>If enabled, the transformer will ignore FIX tags that do not have a corresponding element or attribute in the schema when attempting to transform FIX messages into FIXML. If this attribute is not set then the Node will throw an exception when a corresponding element or attribute cannot be found for a FIX tag in the XSD.</p> <p>Optional. Defaults to <code>false</code>.</p>

### 6.5.3. Customizing FIXML Definitions - FixmlElementGenerator

FIX Protocol Limited (FPL) delivers the FIXML definition in XML Schema Definition (XSD) files. These XSD files come in pairs, one *base* file and one *impl* file. To add a custom fields (as described in FPL's documentation), you must amend the *impl* files. Details of this procedure are contained in the FIXML Schema guide from the [FIX Protocol website](#).

If you need to modify the default schema, please download the FIXML schema from the FPL website and make your modifications.

After the modification you must generate new Java source code based on the new XSD files.

#### FixmlElementGenerator

This tool generates Java code from the FIXML Schema files which is used during the transformation of FIX messages. There is a sample script, *templates/Fixmlv2/FixmlGenerator/<os>/generateFixmlElements.[bat|sh]*, which illustrates the syntax.

There are two Java system properties which must be set in order to run the `FixmlElementGenerator`:

##### 1. `-DresourcesDir`

This should point to the directory containing the XSD files and code templates. There must be subdirectories *xsd* and *fixmltemplates*. Please copy the TEMPLATE files from the jar into *fixmltemplates* and the XSD files into *xsd*. The *xsd* directory must contain subdirectories representing a particular FIX version. The name of these version subdirectories must match the FIX version in a config file where all dots must be replaced by underscores, prefixed with *FIX\_*. So for 5.0SP1 the subdirectory should be named *FIX\_5\_0SP1*



### Note

There are no schema (XSD) files included in the Node. They are available for download from the [FIX Protocol website](#) for all versions.

## 2. -DoutputDir

Point this to any directory you wish the output to be created under. There will also be subdirectories for every version matching the ones from the `resourcesDir`.

When the source code is generated you need to compile the classes (using *catalys-node.jar* in the classpath), pack them into a jar file, change your classpath of the Node to include this jar and at the same time you should not forget about removing the previous FIXML implementation from your classpath.

## 6.5.4. Templates

Refer to *templates/Fixml/Fixml2Fix* for a simple demonstration of how a FIXML message can be transformed to a FIX message.

Refer to *templates/Fixml/Fix2Fixml* for a demonstration of a custom message processor which uses a transformer to print out FIXML messages after they have been transformed from FIX. The source code for the [FixmlTransformPrinter](#) can be found in the *templates/src* directory.

## 6.6. SSL Socket Connection

### 6.6.1. Introduction

Secure Sockets Layer (SSL) and its successor Transport Layer Security (TLS) allow secure, authenticated connections to be made across an insecure network, such as the Internet. SSL and TLS are well-established standard protocols that are widely supported. You can connect to another party, who may be using a FIX engine from a different supplier, with high confidence that the two FIX engines will communicate without difficulty.

SSL/TLS encrypts the whole of the data traffic between the FIX engines, including the protocol and addressing fields, unlike FIX encryption which encrypts only the application data. This makes SSL/TLS unsuitable for systems where an intermediate FIX engine is required to route FIX messages without decrypting them.

In all other situations, SSL/TLS provides the simplest and easiest way of ensuring the security and confidentiality of your FIX messages.

## 6.6.2. Java Keystores and Truststores

The Node uses the standard [Java JSSE](#) (Java Secure Sockets Extension) to provide secure connections between FIX parties. To accept SSL connections on a FIX session, the Node must be configured with a *keystore*; to initiate an SSL connection with another FIX server, the Node must be configured with a *truststore*.

A keystore contains one or more private keys, whereas a truststore contains the public key certificates of those organizations that the user is prepared to trust. JSSE will not permit a client connection to an SSL server unless the certificate presented by the server has been installed in the truststore.

There is no difference in the file format of the keystore and the truststore, and in fact one file can serve as both, though for practical reasons this is not generally recommended. The main reason for this is that the entities you trust are likely to be the same for all of your applications, but the key you use is likely to be different per application.

In most cases, you will need to create and manage your own stores. This is most easily done using the [keytool](#) command-line utility bundled with the Oracle JRE. However, if your counterparty uses a commercial certificate authority to sign its certificates, you may be able to use the default truststore, *cacerts*, that is bundled with the JRE and includes details for several well-known certificate authorities. This file is found in *JAVA\_HOME/jre/lib/security*.

We support usage of standard keystore(PKCS12). PKCS12 keystore file must have the extension ".p12" for it to be recognized as a PKCS12 format file. Truststore, however, cannot be PKCS12 because truststore cannot contain private keys. PKCS12 format is a container for a keypair (private + public key).

## 6.6.3. SSL Protocols

JSSE supports a number of SSL/TLS protocols including TLSv1, TLSv1.1, TLSv1.2, SSLv2Hello and SSLv3.



### Warning

Due to security vulnerabilities in SSLv3, it is recommended that this protocol not be used.

In all newer releases of Oracle's JRE, [SSLv3 is disabled by default](#). If you're running an older release, SSLv3 should be explicitly disabled via the `enabledProtocols` attribute of `SSLSocketConnection`. We recommend setting the value of this attribute to `TLSv1 TLSv1.1 TLSv1.2`, assuming your counterparty supports one of these.

The TLSv1 specification indicates that an SSLv2 "HELLO" message should initiate the handshake so that TLS servers can still talk to v2 servers. Normally the protocol will be negotiated at the highest level of security that both parties support. However some parties may have disallowed, or do not support, the old v2 handshake. In this case you can prevent the Node from sending a v2 HELLO by setting the `noSSLv2Hello` attribute to `true` on the `SSLSocketConnection`. Note that if this is a listening server socket, this will also disallow clients who try to connect using a v2 HELLO, which in most cases is not desirable. Please be sure that you understand the compatibility and security implications before changing this option. Except for the initial handshake, SSL v2 should not be used and is not supported by the Node due to widely known vulnerabilities.

## 6.6.4. Configuration

Configuring the Node for SSL involves two main steps: creating the necessary keystores/truststores, and adding the requisite SSL settings to each applicable session in the Node's configuration file. This process differs depending on whether you are *accepting* SSL connections from other parties or you are *initiating* SSL connections with a counterparty.

A working template that demonstrates the SSL configuration can be found in the *templates/SSL/* directory of your Node installation.

### 6.6.4.1. Configuring a Session to Accept SSL Connections

When accepting SSL connections (you are the SSL server), you must create a keypair and a keystore that contains it. You'll then generate a public key certificate that corresponds to your key; this certificate must be distributed to your clients so that they can configure their system to trust your server.

#### Creating a Keystore and Keypair

To create the keypair in a new keystore:

```
$ keytool -genkeypair -alias fixSSL -keystore server.keystore
Enter keystore password: *****
Re-enter new password: *****
What is your first and last name?
  [Unknown]:  FIX
What is the name of your organizational unit?
  [Unknown]:  FIX
What is the name of your organization?
  [Unknown]:  Company
What is the name of your City or Locality?
  [Unknown]:  Chicago
What is the name of your State or Province?
  [Unknown]:  IL
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=FIX, OU=FIX, O=Company, L=Chicago, ST=IL, C=US correct?
  [no]:  y

Enter key password for <fixSSL>
```



```
(RETURN if same as keystore password):
```

This will generate a new keystore, *server.keystore*, in the current directory. The passwords you choose for the store and the key will be used later when configuring the Node to use this file.

Now the public key certificate must be exported:

```
$ keytool -exportcert -alias fixSSL -keystore server.keystore -file server.cer
Enter keystore password: ***** (the password chosen in the previous step)
Certificate stored in file <server.cer>
```

This will create a PKCS7 file, *server.cer*, which can be distributed to your counterparties. The certificate in this file is *self-signed*; if you wish, you can have your public key signed by a well-known commercial certificate authority, in which case you would provide your counterparty with the certificate(s) created by that CA instead.

It is recommended to keep just a single private key and its associated certificate in any keystore. This is because JSSE by default selects the first private key that it finds in the keystore. You have no control over which private key is selected if there is more than one present.

### Configuring the Session

With the keystore in hand, you can now configure the FIX session to accept SSL connections:

```
<Session compid="SERVER"
  counterpartycompid="CLIENT"
  fixversion="4.2"
  keyStore="/path/to/server.keystore"
  keyStorePassword="keyStorePassword"
  keyPassword="keyPassword">
  <Connections>
    <SSLSocketConnection id="conn" listenport="2000" enabledProtocols="TLSv1.2"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors/>
    <ListenerMessageProcessors/>
  </SessionManager>
</Session>
```

Note we are using an [SSLSocketConnection](#) element instead of the standard `SocketConnection`. The `keyStorePassword` and `keyPassword` attributes of the `Session` must match the passwords you chose when creating the keystore and key in the previous step.

#### 6.6.4.2. Configuring a Session to Initiate an SSL Connection

When initiating SSL connections (you are the SSL client), you must create a truststore that contains the public key certificate of the server to which you are connecting. Your counterparty should have provided this file (or sometimes multiple files).

## Importing the Server's Certificate into a Truststore

In this example, we'll assume the counterparty we are trusting has provided one certificate, *server.cer*. To create a truststore and import this certificate:

```
$ keytool -importcert -alias server -file server.cer -keystore client.truststore
Enter keystore password: *****
Re-enter new password: *****
Owner: CN=FIX, OU=FIX, O=Company, L=Chicago, ST=IL, C=US
Issuer: CN=FIX, OU=FIX, O=Company, L=Chicago, ST=IL, C=US
Serial number: 7b625be9
Valid from: Wed Aug 26 14:57:02 CDT 2015 until: Tue Nov 24 13:57:02 CST 2015
Certificate fingerprints:
...

Trust this certificate? [no]: y
Certificate was added to keystore
```

This will generate a new truststore, *client.truststore*, in the current directory. The password you chose will be used later when configuring the Node to use this file.

The import command asks if you want to trust the imported certificate. In a live SSL environment, you'll be connecting to a remote SSL server, possibly one that belongs to another organization. The owner of that server will have sent you a certificate, most likely via email. How do you know that it hasn't been tampered with while in transit? Every certificate has a *fingerprint*, which `keytool` displays when asking about trust. Before you answer the question, you should contact the person who sent the certificate and have them confirm the fingerprint. This should ideally be done over a different communication channel than the one used to deliver the certificate. After confirming the fingerprint, you can be sure that the certificate has not been interfered with and that it can be safely trusted.

## Configuring the Session

With the truststore in hand, you can now configure the FIX session to use SSL:

```
<Session compid="CLIENT"
  counterpartycompid="SERVER"
  fixversion="4.2"
  trustStore="/path/to/client.truststore"
  trustStorePassword="password">
  <Connections>
    <SSLSocketConnection id="conn" hostname="10.1.1.1" port="2000" enabledProtocols="TLSv1.2"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors/>
    <ListenerMessageProcessors/>
  </SessionManager>
</Session>
```

Note we are using an [SSLSocketConnection](#) element instead of the standard `SocketConnection`. The `trustStorePassword` attribute of the `Session` must match the password you chose when creating the truststore in the previous step.

If your counterparty's SSL certificate was signed by a commercial certificate authority whose certificate already exists in the JRE's `cacerts` truststore, you can omit the `trustStore*` attributes.

## 6.6.5. Client Authentication

In the above configuration scenarios, the initiating FIX session (the SSL client) was configured to trust the server to which it connects. However, the accepting FIX session (the SSL server) had no restrictions on which clients could connect. Theoretically, any client that had the server's public key certificate would be able to initiate a valid SSL connection (assuming they knew the ComplIDs etc).

To create a more comprehensive network of trust, *client authentication* can be used. It provides a way for the server to identify and trust each client, thereby establishing two-way trust. The client is responsible for identifying itself during the connection initiation with the counterparty; the identity provided must match one that the server is configured to trust. The Node can be configured to *require* client authentication when acting as an acceptor, and to *provide* client authentication when acting as an initiator.

### Requiring Client Authentication when Accepting SSL Connections

To configure an acceptor FIX session to require client authentication, both a keystore and a truststore must be present. The keystore holds your server's private key, and the trust store holds the public key certificate(s) of the client(s) you wish to trust and accept.

For this example, we'll enable client authentication for the session we already configured in the [previous section](#). We've already created our keystore; now we must create a truststore that contains the certificates of the clients we trust. We'll assume the client has provided us with its certificate in a file called `clientXYZ.cer`. To create a truststore and import the certificate:

```
$ keytool -importcert -alias clientXYZ -file clientXYZ.cer -keystore server.truststore
Enter keystore password: *****
Re-enter new password: *****
Owner: CN=FIX Client, OU=FIX, O=Client XYZ, L=Chicago, ST=IL, C=US
Issuer: CN=FIX Client, OU=FIX, O=Client XYZ, L=Chicago, ST=IL, C=US
Serial number: 304f4989
Valid from: Thu Oct 01 18:23:09 CDT 2015 until: Wed Dec 30 17:23:09 CST 2015
Certificate fingerprints:
    ...
Extensions:
    ...
Trust this certificate? [no]: yes
Certificate was added to keystore
```

Now the session must be configured to use the truststore and to require clients to identify themselves:

```

<Session compid="SERVER"
  counterpartycompid="CLIENT"
  fixversion="4.2"
  keyStore="/path/to/server.keystore"
  keyStorePassword="keyStorePassword"
  keyPassword="keyPassword"
  needClientAuth="true"
  trustStore="/path/to/server.truststore"
  trustStorePassword="password">
  <Connections>
    <SSLSocketConnection id="conn" listenport="2000" enabledProtocols="TLSv1.2"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors/>
    <ListenerMessageProcessors/>
  </SessionManager>
</Session>

```

With this configuration in place, when a client connects they must present their certificate. If it matches a certificate in the truststore, the connection will commence; otherwise the connection will terminate.

## Providing Client Authentication when Initiating an SSL Connection

As the initiator, a client certificate must be provided when establishing an SSL connection. The process for obtaining a certificate varies depending on the security procedures in place at the counterparty. You may need to generate your own, or the counterparty may provide one for you. To create your own, you'd first generate a keypair, export the public key and then either sign the certificate yourself or have a commercial certificate authority sign it. This can all be accomplished using the `keytool` utility.

In the following example, we'll assume the counterparty has provided everything that's needed, including the private key, *client\_key.pem*, and its corresponding client certificate, *client\_cert.pem*. We'll continue with the session we configured in the [previous section](#), which already has a truststore that includes the certificate of the server. All that's left is to create the keystore:

1. Combine the PEM files into a single PKCS12 file for use with `keytool`:

```

$ openssl pkcs12 -export -in client_cert.pem -inkey client_key.pem -out client.p12 -name client
Enter Export Password:  ***** (choose a password)
Verifying - Enter Export Password:  *****

```

2. Import the PKCS12 file into a Java keystore:

```

$ keytool -importkeystore -srckeystore client.p12 -srcstoretype PKCS12 \
  -destkeystore client.keystore -alias client

Enter destination keystore password:  *****
Re-enter new password:  *****
Enter source keystore password:  ***** (password from previous step)

```

### 3. Configure the session to use the new keystore:

```
<Session compid="CLIENT"
  counterpartycompid="SERVER"
  fixversion="4.2"
  trustStore="/path/to/client.truststore"
  trustStorePassword="password"
  keyStore="/path/to/client.keystore"
  keyStorePassword="password"
  keyPassword="password">
  <Connections>
    <SSLSocketConnection id="conn" hostname="10.1.1.1" port="2000" enabledProtocols="TLSv1.2"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors/>
    <ListenerMessageProcessors/>
  </SessionManager>
</Session>
```

With the keystore in place, Java will automatically provide your client certificate when connecting to a server that requires it.

## 6.6.6. Troubleshooting

The first step in troubleshooting any SSL-related issue is to enable Java SSL debugging, which provides details about the keys that are configured and exchanged during an SSL connection. All of the debug logging is printed to stdout, not to the Catalys log file. To enable, set this system property in your startup command:

```
java -Djavax.net.debug=ssl,handshake,verbose ...
```

Some common SSL issues and their resolutions:

- On startup, an `UnrecoverableKeyException` exception is thrown:

```
java.io.IOException: Error in KeyStore set up
at ...
Caused by: java.security.UnrecoverableKeyException: Cannot recover key
at ...
```

Note there are two separate keystore passwords: one for the store itself, and one for each key therein. This error indicates that the session's `keyPassword` attribute is either missing or does not match the password of the key in the keystore.

- When attempting to initiate an SSL connection, the following error occurs:

```
javax.net.ssl.SSLHandshakeException:
```

```
...
unable to find valid certification path to requested target
```

This indicates that your truststore does not contain a certificate that matches the server's certificate. Enable SSL debugging to see which certificates are configured in your truststore, and which certificate is being presented by the server. If these do not match, consult your counterparty to ensure you have installed the correct certificate.

- When attempting to initiate an SSL connection, the following error occurs:

```
javax.net.ssl.SSLHandshakeException: Received fatal alert: bad_certificate
```

It is likely that your counterparty requires client authentication, but you have not configured a keystore. Please see [Client Authentication](#) for instructions on how to configure this.

- If you would prefer an alternative to the command line keytool, then you can make use of Portecle. This graphical utility can be used to manipulate Java keystores but please see [here](#) for the instructions.

## 6.7. Data Encryption

### 6.7.1. Introduction

On most FIX sessions, security is handled by a secure network provider, and so data encryption is unnecessary. For users who intend to host FIX sessions over the Internet or other open networks, however, security is a prime necessity. Encryption of the session will give you that. By encrypting the sensitive fields and digitally signing each message, both counterparties of a session can ensure relative security that their FIX traffic is being exchanged without intrusion from outside or unwanted parties.

Note that alternatively, encryption of a FIX session can also be achieved with the [SSL Socket Connection](#) option.

The Node can be used with one of several Java encryption packages. However, the binary distribution comes bundled with the freeware [Cryptix](#) cryptography package. As with most Node options, encryption is extensible — you can provide your own encryption engine implementation via the `Encryption` element in your session configuration as described below.

A template can be found in the *templates/Encryption* directory. See the *ReadMe.txt* file in that directory for instructions on running the template.

### 6.7.2. Configuration

Encryption is configured via the `Encryption` element, which is a child of `Session`. It can contain a `PGP-DES-MD5` or a `DES` element. If the `Session` has no `Encryption` element then no encryption will be performed on the session.

## PGP-DES-MD5 Encryption

The following example shows a PGP-DES-MD5 element configured on a session using the given encryption key information. This method uses PGP for Logon exchanges, DES for all messages and MD5 for digital signatures.

For outbound messages, it will also be necessary to specify which FIX fields should be encrypted. In this example, this is done by the `EncryptionSource` message processor, which calls the [setTagSecure\(int, boolean\)](#) method for each field to be encrypted on the outbound `IFIXMessage`. The source code for this processor can be found in *templates/src/com/camerontec/catalys/server/test*.

```
<Session compid="CLIENT" counterpartycompid="SERVER" fixversion="4.2">
  <Encryption>
    <PGP-DES-MD5 class="com.camerontec.catalys.core.crypto.CryptixCrypto"
      publickeyring="../pubring.pgp"
      secretkeyring="../secring.pgp"
      myprivatekeyname="ClientKey"
      passphrase="Client passphrase"
      otherspublickeyname="ServerKey" />
  </Encryption>
  <Persister>
    <JournalingPersister id="jpl" baseDirectoryName="persist/buy" />
  </Persister>
  <Connections>
    <SocketConnection id="conn1" hostname="10.1.1.1" port="2000" />
  </Connections>
  <SessionManager>
    <SourceMessageProcessors>
      <EncryptionSource encrypt.fields="11,21,55,54,38,40,44">
        <SampleMessageSource checkBeforeSend="true" interval="5000" />
      </EncryptionSource>
    </SourceMessageProcessors>
    <ListenerMessageProcessors/>
  </SessionManager>
</Session>
```

For a complete list of configuration attributes, see the [PGP DES MD5 Configuration Reference](#).

## DES Encryption

The following example shows a DES Encryption element configured on a session.

For outbound messages, it will also be necessary to specify which FIX fields should be encrypted. In this example, this is done by the `EncryptionSource` message processor, which calls the [setTagSecure\(int, boolean\)](#) method for each field to be encrypted on the outbound `IFIXMessage`. The source code for this processor can be found in *templates/src/com/camerontec/catalys/server/test*.

```
<Session counterpartycompid="MARKET" compid="BROKER" fixversion="4.4" heartbeat="60">
  <Encryption>
    <DES class="com.camerontec.catalys.core.crypto.CryptixCrypto" />
  </Encryption>
</Session>
```

```

</Encryption>
<Persister>
  <JournalingPersister id="jpl" baseDirectoryName="persist/buy"/>
</Persister>
<Connections>
  <SocketConnection id="conn1" hostname="localhost" port="2000"/>
</Connections>
<SessionManager>
  <SourceMessageProcessors>
    <EncryptionSource encrypt.fields="11,21,55,54,38,40,44,447" id="idN65590">
      <SampleMessageSource interval="15000" checkBeforeSend="true" id="idN65593"/>
    </EncryptionSource>
  </SourceMessageProcessors>
  <ListenerMessageProcessors/>
</SessionManager>
</Session>

```

For a complete list of configuration attributes, see the [DES Configuration Reference](#).

## Logging

All messages are logged in encrypted format. However, all inbound messages reach the message processor later with both decrypted and encrypted data.

### 6.7.3. Encryption Keys

There are two sample programs that show how keys can be generated and put into the PGP key ring files.

1. **PGPKeyGen** - used to generate keys given an encryption key strength (in bits), a passphrase, and a string of characters used to randomize the process of generating the keys.
2. **PGPKeyGenApp** - GUI version of PGPKeyGen.

### 6.7.4. Troubleshooting

Problems encountered usually include the following:

- If the PGP key ring files are not found, you will get a `FileNotFoundException`.
- If the PGP key ring files do not contain the keys declared in the configuration *config.xml* an unknown key exception will be displayed.
- You must also make sure your FIX counterparty is using the same encryption method as configured on your Node for a given session.
- When Cryptix cryptography is being used, some security providers may need to be disabled, as they are not supported by Cryptix. In this case, the following exception will be thrown:

```

(CODE:202) Crypto configuration failed; nested exception is:
    java.security.NoSuchAlgorithmException:

```



```
algorithm RSA/ECB/PKCS1Padding is not available  
from provider Cryptix;
```

In this situation, these security providers must be disabled.

## 6.8. Breakout Box

### 6.8.1. Introduction

The BreakOut feature is a generic mechanism that allows a FIX message to be broken out of normal processing and held in a persistent object, called a [BreakOutBox](#), until the message can be examined and then approved, rejected or resent by a user.



#### Note

The BreakOut option does not provide any user interface for examining/manipulating the contents of a BreakOutBox, it merely provides the infrastructure on which such an interface can be built.

### 6.8.2. Concepts

#### Breaking Out

A message is usually broken out of normal processing because it has failed some sort of check and should not be passed on to the back end system.

For example, a broker might have a message processor that performs client credit limit checks. If an New Order Single message comes in that exceeds the clients limit, then the message processor can break the message out of normal processing and pass it to the BreakOutBox. A user interface can then alert a broker-salesperson, who can then decide whether to:

1. Approve the Order and release.
2. Reject the Order.
3. Increase the client's credit limit and resubmit it to the message processor, to see if it passes any other checks.

#### Related Messages

When a message is broken out there may be subsequent messages that are, in some way, related to the broken out message. If these messages are passed on to the backend system then it will at best

reject them and at worst become confused. Therefore the BreakOutBox can be configured to break out, or reject related messages.

For example, if an New Order Single message has been broken out and then an Order Cancel/Replace Request message comes in for that order, the the Order Cancel/Replace could also be broken out, or it could be rejected.

## Policy Engine

The [BreakOutBox](#) delegates the decisions about related messages to a pluggable [IBreakOutPolicyEngine](#) which is charged with deciding whether incoming message is related to one that is already in the box, and:

1. Should the new message be broken out?
2. Should the message processor stop processing the new message?
3. Should a response message be sent?
4. Should a message that is already in the box be removed?

The policy engine is also in invoked when a message is broken out to decide:

1. Should a response message be sent?
2. Should another message that is already in the box be removed?

## 6.8.3. Examples

This example will demonstrate:

1. How to write a message processor that uses a BreakOutBox box.
2. How to write policy engine.
3. How to configure a BreakOutBox into a Node.

## ExampleBreakOutProcessor

Here is an example message processor, it breaks out Order messages where the OrderQty field is greater than or equal to 1000. The lines in bold are the specific to the BreakOut box; the italicized numbers on the left hand side are referred to in the Notes below:

```
public class ExampleBreakOutProcessor extends FIXSourceListenerBase
{
    private IBreakOutBox breakOutBox;
```

## Optional Features

```
public boolean openComponent()
{
    //1
    setAutoForwarding(false);
    //2
    breakOutBox = (IBreakOutBox)
    ServiceRegistry.getInstance().getService("myBreakOutBox");
    //3
    breakOutBox.registerMessageProcessor(this);
}

public void onMessageFromFix( MessageEvent event, Message message )
{
    //4
    if(breakOutBox.breakOutIfRelated(this, message)) {
        return;
    }
    if(Constants.MSGOrder.equals(message.getMsgType()) &&
        message.getIntegerFieldValue(Constants.TAGiOrderQty) > 1000)
    {
        //5
        breakOutBox.breakOut(this, message, "Order is to big", IBreakOutReason.DEFAULT_CODE);
    }
    else
    {
        //6
        forwardEvent(event);
    }
}
}
```

## Notes

1. Turn off auto forwarding (see [setAutoForwarding\(\)](#) ), because this message processor is going to decide whether to keep processing a message or put it into the break out box.
2. Get a break out box from the [ServiceRegistry](#). This is not always necessary, but it is recommended when the break out box is shared by multiple components (e.g., message processors).
3. Register this message processor with the breakout box. The breakout box needs this so that it can route messages back to the message processor when it breaks them in.
4. When a message arrives, you first need to decide if it is related to some other message in the break out box, then the break out box will take care of it and the message processor does not need to process it any further. E.g., if the message is an OrderCancelReplaceRequest for an Order that already in the break out box, then it could be broken out. [breakOutIfRelated](#) returns `true` if the [BreakOutBox](#) is going to take responsibility for the message.
5. If the message meets some condition (in this case: if it is an order for more 1000 units) then call break it out by calling [breakOut](#)
6. Otherwise forward the message. This is necessary because auto forwarding was turned off in step 1.

## PolicyEngine

The policy engine must be an instance of [IBreakOutPolicyEngine](#) which is an interface with four methods that are called when certain events happen in a [BreakOutBox](#).

The easiest way to write a policy engine is to subclass [AbstractBreakoutPolicyEngine](#) and implement a [AbstractPolicyEngineMessageHandler](#) for each type of message that you handle. A message handler is responsible for making the decisions for one message type. We'll implement message handlers for New Order Single (aka Order) and Order Cancel Request messages.

Here is the policy engine. The full source can be found here: *templates/src/com/camerontec/catalys/server/breakout/policyengine/ExampleBreakOutPolicyEngine.java*.

```
public class ExampleBreakOutPolicyEngine extends AbstractBreakoutPolicyEngine
{
    private Map messageHandlers = new HashMap();

    public ExampleBreakOutPolicyEngine()
    {
        messageHandlers.put(Constants.MSGOrder, new ExampleOrderHandler());
        messageHandlers.put(Constants.MSGOrderCancelRequest, new ExampleOrderCancelHandler());
    }

    protected Map getMessageHandlers()
    {
        return messageHandlers;
    }
}
```

As you can see the policy engine is fairly simple, it's main function is to construct and provide a [Map](#) that maps the message types to message handlers. [AbstractBreakoutPolicyEngine](#) uses this map to select the appropriate message handler for each message.

## ExampleOrderHandler

Here is the handler for order messages.

```
public class ExampleOrderHandler extends AbstractPolicyEngineMessageHandler
{
    private static final int[] TAGS = { Constants.TAGiClOrdID };

    public RelatedMessagePolicyAction determineRelatedMessageAction(
        HandlerContext context,
        IFIXMessageProcessor source,
        Message mess,
        PerSessionIndicies indices)
        throws MessageException
    {
        return new RelatedMessagePolicyAction(false, // don't stopProcessing
            false, // don't breakOut the message
        );
    }
}
```

## Optional Features

```
        null, // no reponseMessage
        null); // don't remove any other message
    }

    public MessageAddedPolicyAction onMessageAdded(HandlerContext context,
        BrokenOutMessageData data,
        PerSessionIndices indices)
        throws MessageException
    {
        return super.onMessageAdded(context, data, indices);
    }

    protected int[] getIDTags()
    {
        return TAGS;
    }
}
```

There are four public methods in a message handler ( [determineRelatedMessageAction\(\)](#), [onMessageAdded\(\)](#), [onMessageRemoved\(\)](#) and [onMessageRecovered\(\)](#) ). They are the same four methods that you can find on the [IBreakOutPolicyEngine](#) interface though they all take a few more parameters. In this case we have implemented the two most interesting ones, although we could have implemented none and had the same effect since we are not doing anything different from the default implementations.

[determineRelatedMessageAction\(\)](#) is called when [breakOutIfRelated\(\)](#) is called by the message processor. It's job is to decide whether the message `mess` is related to some other message in the `BreakOutBox` and if so, what to do about it. In this case, there's nothing to do so we return a `RelatedMessagePolicyAction` that tells the `BreakOutBox` `box` to do nothing. The arguments to the `RelatedMessagePolicyAction` are:

1. `stopProcessing` is the value returned by [breakOutIfRelated\(\)](#), it tells the message processor whether it should stop processing the message.
2. `breakOut` tells the `BreakOutBox` to break out the message.
3. `reponseMessage` is a message to send out in response to this one.
4. `messageToRemove` is a message in the `BreakOutBox` to remove.

[onMessageAdded\(\)](#) is called when the message is added to the `BreakOutBox`. In this case we just call `super.onMessageAdded()` which calls `getIDTags()` to get a list of ID tags for this message, in this case the `ClOrdID` tags, and adds an entry into `indices` for the message that was added (see [PerSessionIndices](#) and [BrokenOutMessageData](#) ) and then returns a [MessageAddedPolicyAction](#) which is a subset of [RelatedMessagePolicyAction](#) instructing the `BreakOutBox` to do nothing more.

The other two methods, [onMessageRemoved\(\)](#) and [onMessageRecovered\(\)](#) are quite similar to [onMessageAdded\(\)](#). [onMessageRemoved\(\)](#) removes the message from the indices. [onMessageRecovered\(\)](#) adds the message, but doesn't return an action.

## ExampleOrderCancelHandler

```

public class ExampleOrderCancelHandler extends AbstractPolicyEngineMessageHandler
{
    public RelatedMessagePolicyAction determineRelatedMessageAction(
        HandlerContext context,
        IFIXMessageProcessor source,
        Message mess,
        PerSessionIndicies indices) throws MessageException
    {
        String origClOrdID = mess.getStringFieldValue(Constants.TAGiOrigClOrdID);

        // Look in the indices, is there a related order?
        BrokenOutMessageData relatedOrderData = indices.getData(Constants.TAGiClOrdID, origClOrdID);
        if (relatedOrderData != null)
        {
            // There is a related order ...
            // Send a Cancelled execution report and remove the Order.
            Message responseMessage = makeOrderCancelledExecutionReport(context, mess);

            return new RelatedMessagePolicyAction(
                true, // do stopProcessing this message (the cancel)
                false, // don't breakout this message
                responseMessage, // send this response
                relatedOrderData); // remove the order from the box
        }
        else
        {
            // It's not related, don't do anything.
            return new RelatedMessagePolicyAction(
                false, // don't stop processing
                false, // don't break out this message
                null, // no response message
                null); // nothing to remove
        }
    }

    private Message makeOrderCancelledExecutionReport(HandlerContext context, Message mess)
        throws MessageException
    {
        Message responseMessage = new Message(Constants.MSGExecution);
        responseMessage.setField(Constants.TAGiOrdStatus, Constants.ORDSTATUS_Cancelled);
        responseMessage.setField(Constants.TAGiExecType, Constants.EXECTYPE_Cancelled);
        responseMessage.setField(Constants.TAGiExecTransType, Constants.EXECTRANSTYPE_New);
        responseMessage.setField(Constants.TAGiClOrdID,
            mess.getStringFieldValue(Constants.TAGiClOrdID));
        responseMessage.setField(Constants.TAGiOrderID,
            context.getOrderIDGenerator().nextId());
        // ...
        return responseMessage;
    }

    protected int[] getIDTags()
    {
        // Don't index this message at all.
        return new int[0];
    }
}

```

```
}
}
```

This class is a little more complicated. `determineRelatedMessageAction()` looks in the indices to see if there is a message already in there with the original `ClOrdId`. If there is, it makes an order cancelled Execution Report, sends it back and removes the original order from the `BreakOutBox`. If there is no message in the indices, then it does nothing. `getIDTags()` returns an empty list, which means that an `OrderCancel` will not be added to the indices.

## 6.8.4. Configuration

A `BreakOutBox` is configured by adding a `BreakOutBox` element into the `Services` element of the `Application`, e.g.

```
<Application>
  <Services>
    <BreakOutBox
      id="myBreakOutBox"
      persistenceFile="bb_persist"
      policyEngineClass=
        "com.camerontec.catalys.server.breakout.policyengine.ExampleBreakOutPolicyEngine"
    />
  </Services>
  ...
</Application>
```

For a complete list of configuration attributes, see the [BreakOutBox Configuration Reference](#).

## 6.9. Market Compliance

### 6.9.1. Introduction

The purpose of the Compliance module in the Node is to enable a broker to enforce a range of trading constraints upon the set of orders received through a FIX connection. These constraints known as "filters" may be a mixture of rules stipulated by market/governmental regulators, internal corporate policies or statutory obligations, or bilaterally-agreed trading rules between the buy/sell sides.

The Compliance module is designed as a message processor conforming to the Catalys architecture, which can be inserted into the FIX processing pipeline to verify compliance with the constraints and provide an opportunity for additional user processing to take place in such circumstances. By default, a failure of any filter will cause the message to be rejected back to the originator with an appropriate FIX message and an exception to be logged. This behavior can be overridden by using the `BreakOut` box which acts as a quarantine area for rejected orders (see [BreakOutBox](#)). Flexible and powerful XML-based rule specification through filter sets, supporting inheritance of filter sets.

The configuration of filters is carried out using XML files.

High-level characteristics of the functionality are:

- Filter sets can be specified differently for each buy side client.
- Filters can be parameterised with different thresholds and limits.
- Support for filter checking by user-specified sub-client definitions.
- Support for configurable price steps for different securities.
- Ability for users to interface code that is automatically notified when filters breach.
- Ability for users to build and deploy their own stateful or stateless filters.
- Hot swappable filter sets.
- Stateful filters operate seamless across system restarts.

### 6.9.2. Compliance Filters

Different compliance constraints are potentially applicable to different types of trading and also to different trading scenarios. They are especially important for automated client order processing (sometimes called direct market access) and automated proprietary order processing. In these cases, the Compliance module would be solely responsible for enforcing a broker's statutory and regulatory obligations. The breach of these is far more serious than the monetary exposure resulting from the orders traded, as it may lead to suspension/withdrawal of their broking license and/or serious financial penalties. It should be noted that brokers are typically fully liable for the actions of their clients on the market.

Each market will typically dictate its own compliance rules. Additionally, there may be a variety of rules, some of which are only applicable to certain types of market activity. A consequence of this is that the Compliance module must be highly configurable, with users being able to select and configure the necessary rules applicable to their situation.

Note that it is not the role of the Compliance module to prevent all trading errors. Rather its focus is on implementing those rules that typically are not able to be picked up by a market trading interface. In some cases, some of the rules may be partly implemented by the market trading interface but this may not offer sufficient flexibility in terms of parameterisation to satisfy regulatory requirements on its own.

The following filters are implemented currently by the Compliance module:

- Check order is no more than x% away (as an absolute value) from market price (best price on opposite side). Captures orders entered in error or those with no likelihood of execution.
- Check order is no more than x% above or below (depending on side) the market price (best price on opposite side). Captures orders entered in error.



## Optional Features

- In markets with an opening and closing auction, then in the pre-trade/closing phases check order is no more than x% away (as an absolute value) from the hypothetical indicative auction price. Captures orders entered in error or those with no likelihood of execution.
- Check order is no more than x% away (as an absolute value) from best bid/ask price (best price on the same side). Captures orders entered in error or those with no likelihood of execution.
- Check order is no more than x% above or below (depending on side) the best bid/ask price (best price on the same side). Captures orders entered in error.
- Check order is no more than x% away (as an absolute value) from the last trade price. Captures orders entered in error or those with no likelihood of execution.
- Check order is no more than x% away above or below (depending on side) the last trade price. Captures orders entered in error.
- Check order is no more than x price steps away (as an absolute value) from market price (best price on opposite side). Captures orders entered in error or those with no likelihood of execution.
- Check order is no more than x price steps above or below (depending on side) the market price (best price on opposite side). Captures orders entered in error.
- In conditions of market overlap (pre-open and closing), check order is no more than x price steps away (as an absolute value) from the hypothetical indicative auction price. Captures orders entered in error or those with no likelihood of execution.
- Check order is no more than x price steps away (as an absolute value) from best bid/ask price (best price on the same side). Captures orders entered in error or those with no likelihood of execution.
- Check order is no more than x price steps above or below (depending on side) the best bid/ask price (best price on the same side). Captures orders entered in error.
- Check order is no more than x price steps away (as an absolute value) from the last trade price. Captures orders entered in error.
- Check order is no more than x price steps above or below (depending on side) from the last trade price. Captures orders entered in error.
- Check order value is no more than the single maximum order value limit for the given client, and no less than the minimum order value limit for the given client. Captures orders entered in error or those with no likelihood of execution.
- Check security is still available for trading. This will check that the security status is in one of a number of after market close on some exchanges leaves an obligation on statuses. Important because for example entering a new order the broker to trade via telephone if requested by a counterparty - something the broker may not want to do for DMA orders.
- Check there are no more than x orders in the market at any one time for a given security for a given client on a particular side of the market. This prevents clients from attempting to give illusion of false market liquidity.

- Check that for a given client and security there are no orders on the opposing side of the market. This prevents the client from trading securities with no beneficial change of ownership.
- Check that security being traded is not one of a list of embargoed securities.
- Check that security type is one of a list of allowed types.
- Check cumulative order value for a given client is no more than certain limit.
- Check cumulative net order value (buys - sells) is no more than certain limit.

### 6.9.3. Operational Model

The Compliance functionality is implemented as a message processor. An instance of the ComplianceProcessor must be configured as both a SourceMessageProcessor and a ListenerMessageProcessor which must be placed before the ASX Trade Transactional gateway. Similarly, as a SourceMessageProcessor, it must sit downstream from the ASX Transactional gateway.

The Compliance processor supports two types of filters - stateful and stateless. A stateless filter is one that only uses the information in the given FIX message to accept or reject the message. A stateful filter will typically use additional state that it accumulates to make the decision. An example of a stateless filter is an order value filter. An example of a stateful filter is a concurrent orders filter.

The Compliance processor will typically only inspect a New Order Single, Order Cancel Replace or Order Cancel Request message on the way into the market. On the way back from the market, stateful filters will inspect any ExecutionReport to update their state.

### 6.9.4. Configuration

The following example shows a ComplianceProcessor element configured on a session that interacts with a [Breakout Box Service](#). If a compliance check fails, then the order message will be sent to the Break Out Box for review.

```
<Application id="ExceptionManagerTemplate">
  <Services>
    <BreakOutBox
      id="exceptionManagerBreakOutBox"
      persistenceFile="./persistence/sell/breakOut"
      policyEngineClass=
        "com.camerontec.catalys.server.breakout.policyengine.ExampleBreakOutPolicyEngine">
      <BreakOutLogger class="com.camerontec.catalys.server.breakout.ExampleBreakOutLogger" />
    </BreakOutBox>
    <JMXManagementService id="JMXManagementService"/>
    <WebServerService id="CatalysWebServerService"/>
  </Services>
  <Sessions>
    <Session counterpartycompid="CLIENT" compid="BROKER" fixversion="4.2" heartbeat="30">
      <Connections>
        <SocketConnection id="idN65566" listenport="3000"/>
      </Connections>
    </Session>
  </Sessions>
</Application>
```

```

</Connections>
<SessionManager>
  <SourceMessageProcessors>
    <ComplianceProcessor id="complianceProcessor"
      rulesFile="../../compliance.txt"
      filterSet="fsl"
      priceStepTableDefinitions="pst.txt"
      exposureFile="./persistence/exposedata"
      marketDataServiceId="marketDataService"
      breakOutBoxId="exceptionManagerBreakOutBox"
      marketMirrorId="sellMirror">
      <SampleExecutor id="executor" />
    </ComplianceProcessor>
  </SourceMessageProcessors>
  <ListenerMessageProcessors>
    <MessageProcessorReference id="complianceProcessorListener" refid="complianceProcessor">
      <MessageProcessorReference id="executorListener" refid="executor" />
    </MessageProcessorReference>
  </ListenerMessageProcessors>
</SessionManager>
</Session>
</Sessions>
</Application>

```

For a complete list of configuration attributes, see the [Compliance Processor Configuration Reference](#).

## 6.9.5. Rules File

Every rules file must conform to the *compliance.dtd* file which is available in the `<CATALYS_NODE>/templates/ExceptionManager` template. The rules file contains one or more compliance filter sets.

Each compliance filter set specifies one or more compliance filters. Additionally, compliance filter sets support inheritance. This allows the specification of a compliance filter set which specialises another. Any compliance filter specified in the specialising filter set supersedes the same filter in the base filter set. Each compliance filter set must have a unique id attribute. When specialising a filter set, the refid attribute is used to specify the base filter set.

The inheritance feature allows client filters to be managed as standard client profiles, with scope for exceptions.

The inheritance feature may be disabled on a filter set family basis by setting the `allowDuplicateFilters` attribute to `true` on the `ComplianceFilterSet` element (default is `false`). In this mode compliance filters in the specialising filter set do not supersede the same filter in the base filter set, but are added in their own right. The end result being that both filters are applied in series. This is useful when you want different instances of the same filter to have different `autoReject` semantics, particularly when this package is used in conjunction with the [Breakout Box](#). For example, it may be desirable to configure two instances of a range type filter with the first having a higher maximum value than the second, and with the first having `autoReject` set to `true` and the second having `autoReject` set to `false`. This effectively creates a 2-tier range filter for which breaches of the upper tier are automatically rejected but breaches of the intermediate tier are broken out for manual authorisation by an operator.

## ConcurrentOrderFilter Attributes

Attributes	Notes
enabled	Sets whether the filter is enabled.  Optional. Default is <code>true</code> .
autoReject	Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.  Optional. Default is <code>true</code> .
tolerance	The maximum number of orders that can exist for a given client/entity for any security and side.  Must be specified.
clientEntityMapper	A class that determines for a given FIX message the entity against which this filter will be applied. This allows the user to perform monitoring at a sub-client level, where the given class performs the user-specific mapping from a message to a sub-client. The class must implement the <code>com.camerontec.catalys.server.compliance.IClientEntityMapper</code> interface. It can derive an entity name from any combination of fields available in the message.  Optional. Default is to monitor based upon client.

## InadvertentCrossingFilter Attributes

Attributes	Notes
enabled	Sets whether the filter is enabled.  Optional. Default is <code>true</code> .
autoReject	Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.  Optional. Default is <code>true</code> .
clientEntityMapper	A class that determines for a given FIX message the entity against which this filter will be applied. This allows the user to perform monitoring at a sub-client level, where the given class performs the user-specific mapping from a message to a sub-client. The class must implement the <code>com.camerontec.catalys.server.compliance.IClientEntityMapper</code>

## Optional Features

Attributes	Notes
	<p>interface. It can derive an entity name from any combination of fields available in the message.</p> <p>Optional. Default is to monitor based upon client.</p>

## GrossExposureFilter Attributes

Attributes	Notes
enabled	<p>Sets whether the filter is enabled.</p> <p>Optional. Default is <code>true</code>.</p>
autoReject	<p>Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.</p> <p>Optional. Default is <code>true</code>.</p>
tolerance	<p>The maximum number of orders that can exist for a given client/entity for any security and side.</p> <p>Must be specified.</p>
clientEntityMapper	<p>A class that determines for a given FIX message the entity against which this filter will be applied. This allows the user to perform monitoring at a sub-client level, where the given class performs the user-specific mapping from a message to a sub-client. The class must implement the <code>com.camerontec.catalys.server.compliance.IClientEntityMapper</code> interface. It can derive an entity name from any combination of fields available in the message.</p> <p>Optional. Default is to monitor based upon client.</p>

## NetExposureFilter Attributes

Attributes	Notes
enabled	<p>Sets whether the filter is enabled.</p> <p>Optional. Default is <code>true</code>.</p>
autoReject	<p>Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.</p>

## Optional Features

Attributes	Notes
	Optional. Default is <code>true</code> .
tolerance	<p>The maximum number of orders that can exist for a given client/entity for any security and side.</p> <p>Must be specified.</p>
clientEntityMapper	<p>A class that determines for a given FIX message the entity against which this filter will be applied. This allows the user to perform monitoring at a sub-client level, where the given class performs the user-specific mapping from a message to a sub-client. The class must implement the <code>com.camerontec.catalys.server.compliance.IClientEntityMapper</code> interface. It can derive an entity name from any combination of fields available in the message.</p> <p>Optional. Default is to monitor based upon client.</p>

## GrossTradeExposureFilter Attributes

Attributes	Notes
enabled	<p>Sets whether the filter is enabled.</p> <p>Optional. Default is <code>true</code>.</p>
autoReject	<p>Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.</p> <p>Optional. Default is <code>true</code>.</p>
tolerance	<p>If the trade value is less then specified tolerance orders can be submitted by a single client/entity. Tolerance is expressed as a gross value. Amends to reduce order value are permitted.</p> <p>Must be specified.</p>
clientEntityMapper	<p>A class that determines for a given FIX message the entity against which this filter will be applied. This allows the user to perform monitoring at a sub-client level, where the given class performs the user-specific mapping from a message to a sub-client. The class must implement the <code>com.camerontec.catalys.server.compliance.IClientEntityMapper</code> interface. It can derive an entity name from any combination of fields available in the message.</p>

Attributes	Notes
	Optional. Default is to monitor based upon client.

## NetTradeExposureFilter Attributes

Attributes	Notes
enabled	Sets whether the filter is enabled.  Optional. Default is <code>true</code> .
autoReject	Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.  Optional. Default is <code>true</code> .
tolerance	If the trade value is less then specified tolerance orders can be submitted by a single client/entity. Tolerance is expressed as a net value.  Must be specified.
clientEntityMapper	A class that determines for a given FIX message the entity against which this filter will be applied. This allows the user to perform monitoring at a sub-client level, where the given class performs the user-specific mapping from a message to a sub-client. The class must implement the <code>com.camerontec.catalys.server.compliance.IClientEntityMapper</code> interface. It can derive an entity name from any combination of fields available in the message.  Optional. Default is to monitor based upon client.

## MarketModeFilter Attributes

Attributes	Notes
enabled	Sets whether the filter is enabled.  Optional. Default is <code>true</code> .
autoReject	Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.  Optional. Default is <code>true</code> .
allowed	A comma-separated list of market modes that are allowed.

Attributes	Notes
	Must be specified.

## OrderValueFilter Attributes

Attributes	Notes
enabled	Sets whether the filter is enabled.  Optional. Default is <code>true</code> .
autoReject	Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.  Optional. Default is <code>true</code> .
maxNormal	The maximum value of an order that can be sent in the Normal phase.  Must be specified.
maxPreOpen	The maximum value of an order that can be sent in the PreOpen phase.  Must be specified.
minNormal	The minimum value of an order that can be sent in the Normal phase.  Optional. Defaults to 0.
minPreOpen	The minimum value of an order that can be sent in the PreOpen phase.  Optional. Defaults to 0.

## PercentageFromAuctionPriceFilter Attributes

Attributes	Notes
enabled	Sets whether the filter is enabled.  Optional. Default is <code>true</code> .
autoReject	Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.  Optional. Default is <code>true</code> .



## Optional Features

Attributes	Notes
tolerance	<p>The maximum number of orders that can exist for a given client/entity for any security and side.</p> <p>Must be specified.</p>

## PercentageFromBestPriceFilter Attributes

Attributes	Notes
enabled	<p>Sets whether the filter is enabled.</p> <p>Optional. Default is <code>true</code>.</p>
autoReject	<p>Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.</p> <p>Optional. Default is <code>true</code>.</p>
maxNormal	<p>The maximum permissible percentage away from the best bid/ask price in Normal phase of the order limit price. If there is no best bid/ask price, the filter succeeds.</p> <p>Must be specified.</p>
maxPreOpen	<p>The maximum permissible percentage away from the best bid/ask price in PreOpen phase of the order limit price. If there is no best bid/ask price, the filter succeeds.</p> <p>Must be specified.</p>
absolute	<p>Whether the filter should measure the absolute or relative difference in prices.</p> <p>Default is <code>false</code>.</p>

## PercentageFromLastPriceFilter Attributes

Attributes	Notes
enabled	<p>Sets whether the filter is enabled.</p> <p>Optional. Default is <code>true</code>.</p>

## Optional Features

Attributes	Notes
autoReject	Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.  Optional. Default is <code>true</code> .
maxNormal	The maximum permissible percentage away from the last trade price in Normal phase of the order limit price expressed as an absolute value. If there is no last trade price and <code>breachIfNoReferencePrice</code> is false, the filter succeeds.  Must be specified.
maxPreOpen	The maximum permissible percentage away from the last trade price in pre-open phase of the order limit price expressed as an absolute value. If there is no last trade price and <code>breachIfNoReferencePrice</code> is false, the filter succeeds.  Must be specified.
useAuctionIfNoLast	If set to true and there is no last price indicative auction price is used as reference price.  Optional. Default is <code>false</code> .
breachIfNoReferencePrice	If set to true and there is no last price the filter breaches.  Optional. Default is <code>false</code> .

## PercentageFromMarketPriceFilter Attributes

Attributes	Notes
enabled	Sets whether the filter is enabled.  Optional. Default is <code>true</code> .
autoReject	Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.  Optional. Default is <code>true</code> .
maxNormal	The maximum permissible percentage away from the market price in normal phase of the order limit price expressed as an absolute value. The market

## Optional Features

Attributes	Notes
	price is defined to be the best price on the opposing side of the market. If there is no market price, the filter succeeds.  Must be specified.
maxPreOpen	The maximum permissible percentage away from the market price in preopen phase of the order limit price expressed as an absolute value. The market price is defined to be the best price on the opposing side of the market. If there is no market price, the filter succeeds.  Must be specified.

### PriceStepsFromAuctionPriceFilter Attributes

Attributes	Notes
enabled	Sets whether the filter is enabled.  Optional. Default is <code>true</code> .
autoReject	Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.  Optional. Default is <code>true</code> .
tolerance	The maximum permissible price steps away from the auction price of the order limit price expressed as an absolute value. If there is no auction price, the filter succeeds.  Must be specified.

### PriceStepsFromBestPriceFilter Attributes

Attributes	Notes
enabled	Sets whether the filter is enabled.  Optional. Default is <code>true</code> .
autoReject	Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.  Optional. Default is <code>true</code> .

## Optional Features

Attributes	Notes
maxNormal	<p>The maximum permissible price steps away from the best bid/ask price in normal phase of the order limit price expressed as an absolute value. If there is no best bid/ask price, the filter succeeds.</p> <p>Must be specified.</p>
maxPreOpen	<p>The maximum permissible price steps away from the best bid/ask price in preopen phase of the order limit price expressed as an absolute value. If there is no best bid/ask price, the filter succeeds.</p> <p>Must be specified.</p>

## PriceStepsFromLastPriceFilter Attributes

Attributes	Notes
enabled	<p>Sets whether the filter is enabled.</p> <p>Optional. Default is <code>true</code>.</p>
autoReject	<p>Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.</p> <p>Optional. Default is <code>true</code>.</p>
maxNormal	<p>The maximum permissible price steps away from the last trade price in normal phase of the order limit price expressed as an absolute value. If there is no last trade price and <code>breachIfNoReferencePrice</code> is false, the filter succeeds.</p> <p>Must be specified.</p>
maxPreOpen	<p>The maximum permissible price steps away from the last trade price in preopen phase of the order limit price expressed as an absolute value. If there is no last trade price and <code>breachIfNoReferencePrice</code> is false, the filter succeeds.</p> <p>Must be specified.</p>
absolute	<p>Whether the filter should measure the absolute or relative difference in prices.</p> <p>Default is <code>false</code>.</p>

## Optional Features

Attributes	Notes
useAuctionIfNoLast	<p>If set to true and there is no last price indicative auction price is used as reference price.</p> <p>Optional. Default is <code>false</code>.</p>
breachIfNoReferencePrice	<p>If set to true and there is no last price the filter breaches.</p> <p>Optional. Default is <code>false</code>.</p>

## PriceStepsFromMarketPriceFilter Attributes

Attributes	Notes
enabled	<p>Sets whether the filter is enabled.</p> <p>Optional. Default is <code>true</code>.</p>
autoReject	<p>Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.</p> <p>Optional. Default is <code>true</code>.</p>
maxNormal	<p>The maximum permissible price steps away from the market price in normal phase of the order limit price expressed as an absolute value. The market price is defined to be the best price on the opposing side of the market. If there is no market price, the filter succeeds.</p> <p>Must be specified.</p>
maxPreOpen	<p>The maximum permissible price steps away from the market price in preopen phase of the order limit price expressed as an absolute value. The market price is defined to be the best price on the opposing side of the market. If there is no market price, the filter succeeds.</p> <p>Must be specified.</p>
absolute	<p>Whether the filter should measure the absolute or relative difference in prices.</p> <p>Default is <code>false</code>.</p>

## SecurityFilter Attributes

Attributes	Notes
enabled	Sets whether the filter is enabled.  Optional. Default is <code>true</code> .
autoReject	Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.  Optional. Default is <code>true</code> .
disallowed	A comma-separated list of securities in which orders are disallowed. These should be specified in market specific codes, e.g. for the Catalys ASX Trade market, ASX Trade codes should be used.  Optional. By default all security types are allowed.
absolute	Whether the filter should measure the absolute or relative difference in prices.  Default is <code>false</code> .

## SecurityTypeFilter Attributes

Attributes	Notes
enabled	Sets whether the filter is enabled.  Optional. Default is <code>true</code> .
autoReject	Whether a failure of this filter will cause automatic rejection. If false, then the breach will be delegated to the Breakout Box for trader attention.  Optional. Default is <code>true</code> .
disallowed	A comma-separated list of security types that are disallowed. Note security types are market specific.  Optional. By default all security types are allowed.
absolute	Whether the filter should measure the absolute or relative difference in prices.  Default is <code>false</code> .

## 6.9.6. Price Bands for Percentage Filters

For all percentage price filters different tolerances can be configured for different price ranges. The ability to differentiate between normal market operation and pre-open phase will remain. It is also possible to specify global price tolerance tables which can be referenced from a filter as well as define filter specific tables. It applies to the following filters:

- PercentageFromAuctionPriceFilter
- PercentageFromBestPriceFilter
- PercentageFromLastPriceFilter
- PercentageFromMarketPriceFilter

### PriceToleranceTable

Attributes	Notes
id	Unique identifier  Must be specified.
<PriceToleranceTableEntry>	<PriceToleranceTable> can contain an unlimited amount of <PriceToleranceTableEntry> elements as children in which the tolerance for each price range is defined.  Must be specified.

### PriceToleranceTableEntry

Attributes	Notes
toPrice	The upper limit of the price for which the tolerance applies. The start of the price range is the <i>toPrice</i> of the previous entry (or 0 if the first one). A blank value indicates infinity.  Must be specified.
maxPreOpen	Value for the tolerance at this price range while the market is in pre-open Phase.  Must be specified.
maxNormal	Value for the tolerance at this price range during normal trading operation.

Attributes	Notes
	Must be specified.

## PriceToleranceTableReference

This element can only be used as a child for a percentage filter.

Attributes	Notes
refid	Referencing an id of a <i>&lt;PriceToleranceTable&gt;</i> which is defined globally, i.e. outside a filter element and under <i>&lt;ComplianceFilterSets&gt;</i>  Must be specified.

## Configuration Example

```
<?xml version="1.0"?>
<!DOCTYPE ComplianceFilterSets SYSTEM "compliance.dtd">
<ComplianceFilterSets> + " "
  <PriceToleranceTable id="pttr">
    <PriceToleranceTableEntry toPrice="0.008" maxNormal="100" maxPreOpen="500"/>
    <PriceToleranceTableEntry toPrice="0.04" maxNormal="20" maxPreOpen="200"/>
    <PriceToleranceTableEntry toPrice="" maxNormal="10" maxPreOpen="50"/>
  </PriceToleranceTable>
  <ComplianceFilterSet id="fsl">
    <PercentageFromAuctionPriceFilter>
      <PriceToleranceTable id="ptt-pfapf">
        <PriceToleranceTableEntry toPrice="0.006" maxNormal="200" maxPreOpen="200"/>
        <PriceToleranceTableEntry toPrice="0.02" maxNormal="50" maxPreOpen="50"/>
        <PriceToleranceTableEntry toPrice="" maxNormal="25" maxPreOpen="25"/>
      </PriceToleranceTable>
    </PercentageFromAuctionPriceFilter>
    <PercentageFromBestPriceFilter>
      <PriceToleranceTable id="ptt">
        <PriceToleranceTableEntry toPrice="0.006" maxNormal="200" maxPreOpen="1000"/>
        <PriceToleranceTableEntry toPrice="0.02" maxNormal="50" maxPreOpen="200"/>
        <PriceToleranceTableEntry toPrice="" maxNormal="25" maxPreOpen="100"/>
      </PriceToleranceTable>
    </PercentageFromBestPriceFilter>
    <PercentageFromLastPriceFilter>
      <PriceToleranceTableReference refid="ptt" />
    </PercentageFromLastPriceFilter>
    <PercentageFromMarketPriceFilter>
      <PriceToleranceTableReference refid="pttr" />
    </PercentageFromMarketPriceFilter>
  </ComplianceFilterSet>
</ComplianceFilterSets>
```



## 6.9.7. Price Step Table File

The Price Step Table file defines price steps for securities. Every Price Step Table file must conform to a price step element located in the *Compliance.dtd* file. The mapping is performed on the basis of a price step table identifier. Each security is assumed to be inherently associated with a price step table identifier.

The price step table file contains a number of price step definitions. These effectively define a step function. Each price step table is identified with a unique id. Each price step table has one or more price step table entries. These define the discontinuities in the step function.

The following section describes the various attributes that can be specified on a PriceStepTableEntry.

### PriceStepTableEntry Attributes

Attributes	Notes
maxPrice	The value below which the price step is the given priceStep value.  Must be specified.
priceStep	Size of the price step.  Must be specified.
absolute	Whether the filter should measure the absolute or relative difference in prices.  Default is <code>false</code> .

Note that the price step table entries are automatically ordered when being processed. Also note that any value above the highest maxPrice will have a priceStep of the highest maxPrice price step table entry.

## 6.9.8. Extending Filters

Users may choose to write additional filters based upon existing or other information available. There are two types of filters as discussed earlier, stateful and stateless. Both must provide an implementation of the `com.camerontec.catalys.server.compliance.ComplianceFilter` class. This class should perform necessary checks in the `validate` method and throw an exception if a breach is detected.

Additionally, stateful filters may want to extend the `com.camerontec.catalys.server.compliance.AbstractFilterStateController` class. This class is invoked on messages from the exchange and can be used to correctly maintain any state.

Users will need to add their filters to the *compliance.dtd* file to define filter attributes and update the rules file(s) with relevant config.

When the Compliance module starts, it will automatically read the filter configuration from the rules file, and instantiate all filters and filter state controllers.

### 6.9.9. The `absolute` Attribute

Some filters which measure the difference between an orders price and some reference price have a boolean attribute called `absolute`. If this attribute to `true` then the difference is measured as an absolute value. E.g., the filter is breached if:

```
abs(order_price - reference_price) > tolerance
```

where `order-price` is the price taken from the order (or order amend) message, `reference-price` is the reference price (e.g, best price, last price or market price), `tolerance` is the maximum difference and `abs` is the absolute value function. (Note: some filters express these values as price steps or percentages).

If the `absolute` attribute is `false` then the difference is relative and depends on the side of the order. E.g., for buy orders the filter is triggered if:

```
(order_price - reference_price) > tolerance
```

For sell orders the filter is triggered if:

```
(reference_price - order_price) > tolerance
```

## 6.10. Message Compression

### 6.10.1. FAST Support

To configure the Catalys Node Server to use FAST:

- Place a FAST template file in a known location.
- Add a `Compression` element to your configuration. The `templateFile` attribute of the FAST sub-element must point to your template file.
- Add `toFIXCompressionRefId` and `fromFIXCompressionRefId` attributes to the `Session` elements that are required to use FAST.

See the templates located in *templates/Fast* directory.

Sessions configured to use FAST will:

- Encode outgoing messages and decode incoming messages using FAST.
- Use a `FastFriendlyFIXMessageFactory` to create `IFIXMessage` objects that are optimized for FAST.
- The `printMessages` attribute will produce additional information. This information can give you an insight into the FAST encoding/decoding process.

## 6.10.2. FAST and the FIX Libraries

The Catalys Node Libraries can be programmatically configured to use FAST when they are not used within the larger framework of the Node. The Session API contains set and get methods for the [SessionFastConfigurationHelper](#) class. This class is used to construct and configure the session's inbound and outbound FAST configuration.

So that messages of the most efficient type are created, it is also advisable to set the message factories of the session to [FastFriendlyFIXMessageFactory](#). Message factories of the session are set by the following methods:

- [Session.setFromFIXMessageFactory\(IFIXMessageFactory\)](#)
- [Session.setToFIXMessageFactory\(IFIXMessageFactory\)](#)

Sample code demonstrating how to create FAST server and client(s) can be found in the [FastServerExample](#) and [FastClientExample](#) classes located in the *templates/src* directory. These template classes are used as part of the FastServer and FastClient templates found in the *templates/Fast/Core* directory of the Node distribution. The same example shows how to setup the client/server code to use the FAST Session Control Protocol (SCP).

Sample code demonstrating how to create `FastInputStream` and `FastOutputStream` objects can be found in the [FastEncoder](#) and [FastDecoder](#) classes located in the *templates/src* directory. These template classes are used as part of the FastEncoder and FastDecoder templates found in the *templates/Fast/Encoding* directory of the Catalys Node distribution. Note that these templates show how to use FAST in a stand-alone manner, not in an integrated manner as described here.

## 6.10.3. Additional Notes

- If there is more than one `SocketConnection` configured to listen on the same port, they must all be attached to sessions with the same compression configuration.
- The Node FAST libraries use the `id` attribute of FAST templates to associate each template with a FIX `MsgType`. This means that each FAST message template must contain a `MsgType` field whose value is constant and corresponds to the FIX message type. The `id` of each template must also be unique.
- The session dialect [dialectNumericSendingTime](#) allows sending times to be represented as numeric values. This allows the Node to handle FAST templates that define sending times using numeric field types.

## 6.11. Market Data Services

### 6.11.1. Introduction

A Node Market Data Service (MDS) is a Java object that knows how to process market data (e.g. prices) coming from a particular source. All stock exchanges provide some kind of a price feed giving the current

prices of stocks traded in that exchange. They are a common source of market data. Market data can also come from third parties such as Reuters.

Typically, each exchange or third party provides market data in their own proprietary format. The job of a Node MDS is to interpret the market data coming from a particular source and make that data available in a standard form which can be used by Node components. Node users can also access the market data in their own code (e.g. in a custom message processor).

## 6.11.2. Configuring a Market Data Service

Node MDS are a special kind of Node service. Every Node service must implement our standard [IService](#) interface. Market data services must implement [IMarketDataService](#) (which extends [IService](#)).

MDS are configured by specifying them under the `Services` element. Like all services, each service is assigned a unique id which becomes the name by which the service is known.

Most MDS inherit from [MarketDataServiceBase](#). All such services support an optional nested `NonStandardData` element which allows for non standard FIX market data fields. The `NonStandardData` element can contain multiple `MarketDataField` elements. The `MarketDataField` element has the following attributes:

### MarketDataField Attributes

Attributes	Notes
name	Name of field.  Required.
entryType	Value of FIX MDEntryType (tag 269) corresponding to this field.  Required for fields in FIX MarketDataFullRefresh and MarketDataIncrementalRefresh messages (MsgTypes W and X).  It is not required for fields that map directly to a unique FIX tag.
fixTag	FIX Tag corresponding to field.
securityStatus	<code>true</code> if this field should go out in a FIX SecurityStatus message (MsgType f).  <code>false</code> if this field should go out in FIX MarketDataFullRefresh and MarketDataIncrementalRefresh messages (MsgTypes W and X).  Optional. Defaults to <code>false</code> .

Attributes	Notes
isPrice	<p>true if this field is price type.</p> <p>false if this field is not price type.</p> <p>Optional. Defaults to false.</p>

## Configuration Example

```
<Services>
  <InternalMarketDataService id="fixMarketData">
    <NonStandardData>
      <MarketDataField name="MyVwapPrice" entryType="A" fixTag="270" />
    </NonStandardData>
  </InternalMarketDataService>
</Services>
```

### 6.11.3. Using a Market Data Service

The job of an `IMarketDataService` object is to read market data from some source and update its internal store of current market data. So every MDS has an internal market data store. It can be retrieved from a market data service by calling the service's [getMarketDataStore](#) method. This returns the market data as an [IMarketDataStore](#) object.

An [IMarketDataStore](#) is basically a collection of [IInstrument](#) objects. Each [IInstrument](#) object represents an instrument (e.g. stock) for which we have market data. Other objects can access instrument data by calling the [getInstrument](#) method.

Other objects can also register themselves as listeners to the market data store, in which case they will be notified of market data changes. See the various add listener methods of `IMarketDataStore` and the interfaces which listening objects must implement.

See the JavaDoc for the above classes for more detailed information.

### 6.11.4. Writing a Market Data Service

The full source code for a sample market data service, [SampleMarketDataService](#), can be found at `templates/src/com/camerontec/catalys/server/marketdata/service/SampleMarketDataService.java`.

Almost all of the code in a MDS involves reading and interpreting the source of market data that your MDS wants to process. Once you have extracted some market data, (e.g. a price update for a particular stock), you need to update your internal [IMarketDataStore](#) (see [previous section](#)). Fortunately, you will not normally need to write your own `IMarketDataStore` and `IInstrument` implementations. You should be able to use the standard implementations provided by [StandardMarketDataStore](#) and [StandardInstrument](#).

Note that you can add instruments to `StandardMarketDataStore` using [addInstrument](#). Note also that you can update the market data associated with a [StandardInstrument](#) using various set methods such as [setTopOfBook](#).

All MDS declared in the `Services` configuration element are instantiated when the Node is started. MDS comply to the [IComponent](#) interface. After instantiation of the service object, the [openComponent](#) method will be called. This is where your service should read its configuration attributes (e.g., using the [getAttribute](#) method) and configure itself. The service should only become active (eg by actually connecting and subscribing to a data source) when the [setComponentActive](#) method is called with parameter *true*.

### 6.11.5. Writing a Derived Market Data Service

A *derived* market data service is a market data service which depends on one or more other underlying MDS.

Instead of reading and processing incoming data from some external market data source, a derived MDS process locates the underlying market data services that it depends on and registers itself as a listener to their `IMarketDataStores`. It processes the updates coming in from those underlying stores and updates its own `IMarketDataStore` with the result of the processing.

There are two templates of derived market data services:

- [SampleDerivedPriceMarketDataService](#). Full source code can be found at `templates/src/com/camerontec/catalys/server/marketdata/test/SampleDerivedPriceMarketDataService.java`.
- [SampleAggregatedMarketDataService](#). Full source code can be found at `templates/src/com/camerontec/catalys/server/marketdata/test/SampleAggregatedMarketDataService.java`.

# Chapter 7. Technology Adapters

## Table of Contents

7.1. Socket Adapter .....	230
7.1.1. Introduction .....	230
7.1.2. Socket Adapter Protocol .....	231
7.1.3. Configuration .....	233
7.1.4. Sample Client Code .....	234
7.1.5. Message Transformers .....	236
7.1.6. Multi-Session Support .....	236
7.1.7. Message Acknowledgements .....	237
7.2. Advanced Socket Adapter .....	237
7.2.1. Simple vs. Advanced .....	237
7.2.2. Application Architecture .....	238
7.2.3. Protocol .....	238
7.2.4. Configuration .....	251
7.2.5. Command-Line Operations .....	252
7.3. Embedded Catalys Node Adapter .....	253
7.3.1. Introduction .....	253
7.3.2. Adapter Operation .....	253
7.3.3. Configuration .....	253
7.3.4. Templates and Example Client Code .....	254
7.3.5. Performance .....	256
7.4. JMS Adapter .....	256
7.4.1. Introduction .....	256
7.4.2. JMS Provider Installation .....	256
7.4.3. Configuration .....	257
7.4.4. Templates .....	258
7.5. Solace Adapter .....	258
7.5.1. Introduction .....	258
7.5.2. Installation .....	258
7.5.3. Functionality .....	259
7.5.4. Solace Adapter Components .....	259
7.5.5. Configuration .....	262
7.5.6. Performance Tuning .....	264
7.5.7. Using the Adapter with Catalys Node High Availability .....	265
7.5.8. JCSMP API Logging .....	266
7.6. IBM MQSeries Adapter .....	266
7.6.1. Introduction .....	266
7.6.2. MQSeries Installation .....	266

7.6.3. Typical Configuration and Message Conversion .....	266
7.6.4. Configuration .....	268
7.7. TIBCO Rendezvous Adapter .....	270
7.7.1. Introduction .....	270
7.7.2. Adapter Installation .....	270
7.7.3. Typical Configuration and Message Conversion .....	270
7.7.4. Configuration .....	271
7.7.5. Templates .....	273
7.8. Microsoft MQ Adapter .....	274
7.8.1. Introduction .....	274
7.8.2. Overview .....	274
7.8.3. MSMQ Installation .....	275
7.8.4. Configuration .....	275
7.8.5. Troubleshooting .....	276
7.9. JDBC Adapter .....	277
7.9.1. Overview .....	277
7.9.2. JDBC Adapter Installation .....	277
7.9.3. Configuration .....	277
7.9.4. Template .....	278
7.9.5. Repeating Groups Considerations .....	279
7.10. Flat File Adapter .....	279
7.10.1. Introduction .....	279
7.10.2. Operation .....	279
7.10.3. Configuration .....	280
7.10.4. File System Persistent Message Queue .....	280
7.10.5. File Naming Conventions .....	282
7.10.6. Queue Snapshots .....	283
7.10.7. Templates .....	284
7.11. RMI Adapter .....	284
7.11.1. Introduction .....	284
7.11.2. Configuration .....	284
7.11.3. Catalys Node Object Model .....	286
7.11.4. Class Summary .....	286
7.11.5. Templates .....	287
7.11.6. Troubleshooting .....	288
7.12. EMX Adapter .....	289
7.12.1. Introduction .....	289
7.12.2. Introducing the EMX Integrator .....	290
7.12.3. Installation .....	291
7.12.4. Running the Demonstration Programs .....	291
7.12.5. Developing Your System .....	297
7.12.6. Performing the EMX Compliance Tests .....	301
7.12.7. Obtaining a Certificate for the Live EMX System .....	303



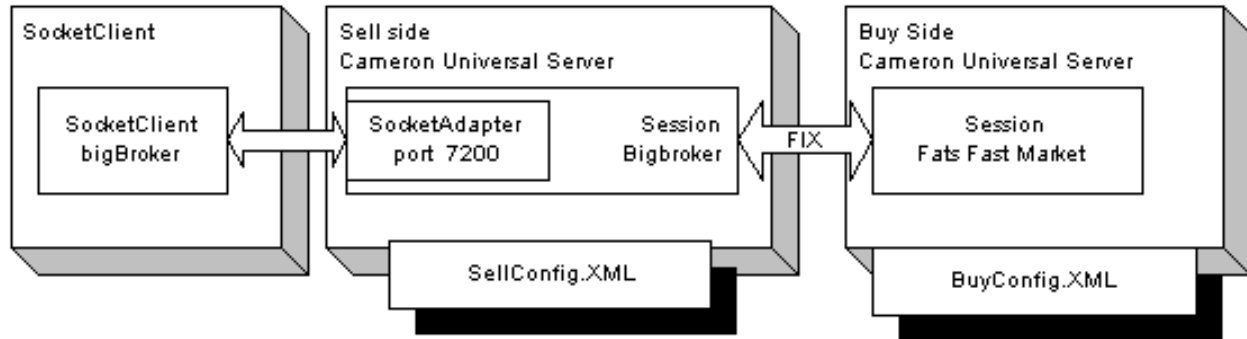
7.12.8. Other Information .....	304
7.13. EMX Delimited File Adapter .....	304
7.13.1. Introduction .....	304
7.13.2. Configuration and Initial Set Up .....	305
7.13.3. EMX Delimited File Adapter User Guide .....	305
7.13.4. Daily Running .....	307
7.13.5. Outgoing Message Files .....	308
7.13.6. Incoming Message Files .....	311
7.13.7. Operational Issues .....	313
7.14. C/C++ API .....	314
7.14.1. Introduction .....	314
7.14.2. Overview .....	314
7.14.3. Programming Notes .....	315
7.14.4. Library .....	315
7.14.5. Class Diagram .....	316
7.14.6. Sequence Diagram .....	320
7.14.7. Templates .....	321
7.14.8. Sample user code .....	321
7.15. Microsoft .NET API .....	322
7.15.1. Introduction .....	322
7.15.2. Installation .....	323
7.15.3. Getting Started .....	323
7.15.4. The Catalys Node .NET API .....	323
7.15.5. Catalys Node Configuration .....	327

## 7.1. Socket Adapter

### 7.1.1. Introduction

The Socket Adapter allows non-FIX applications to connect to the Node in order to send and receive FIX messages. It uses a simple socket messaging format. This technology adapter normally supports applications written in Java, C/C++ or .NET.

The architecture of a Node running with a socket adapter typically looks like this:



## 7.1.2. Socket Adapter Protocol

Socket clients and the socket adapter use a well-defined messaging protocol to connect and pass messages, data and events between each other. Message acknowledgement is also supported between client and adapter.

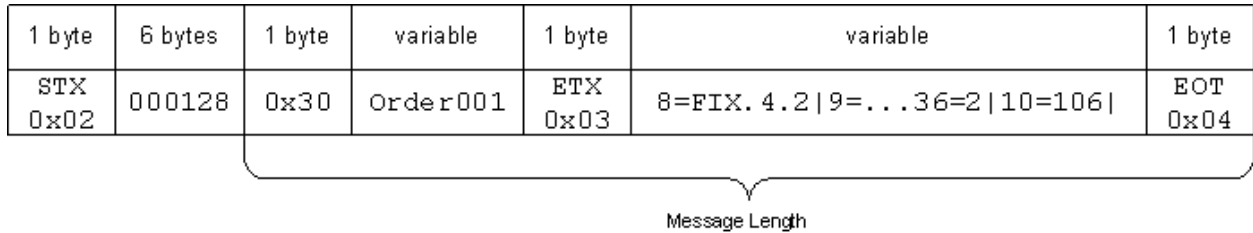
### Message Format

The messaging protocol uses a message format defined in the following table:

Field Name	Size	Description
STX 0x02 (Start Of Text)	1 byte	Indicator for the start of the message.
MessageLength	6 or 9 bytes	The total length of the entire message excluding the STX and MessageLength field itself. This field is represented either as a 6 or 9 char text string padded with leading zeros (e.g. "000128"). The size of this field depends on whether the <code>largeMessages</code> attribute has been set.
MessageType	1 byte	An 8 bit number that identifies the type of the message. See the next table for a list of supported message types.
MessageId	variable	A unique identifier created by the client program so that any outgoing messages can easily be acknowledged.
ETX 0x03 (End Of Text)	1 byte	Because the MessageId can be any length this byte indicates the end of the MessageId field.
Data	variable	

Field Name	Size	Description
EOT 0x04 (End Of Trans)	1 byte	

The following diagram shows a typical socket message. Note the area the MessageLength covers.



## Message Types

Supported message types:

MessageType	ID	Description
SEND_MESSAGE	0x30	Message used to send an outbound FIX messages from the Node. The Node will respond with either a <code>RECV_ACK</code> or <code>RECV_NACK</code> depending on whether or not the message was sent. A unique <code>MessageId</code> must be set..
RECV_MESSAGE	0x31	Message received containing an inbound FIX message.
RECV_ACK	0x32	Message received when an outbound message is valid and has been sent using a <code>SEND_MESSAGE</code> . The <code>MessageId</code> field maps to the original value in the <code>SEND_MESSAGE</code> . The data field is empty.
RECV_NACK	0x33	Message received when a message fails to be sent out a FIX session. The <code>MessageId</code> field maps to the original value in the <code>SEND_MESSAGE</code> . The data field contains a description of why the message failed to be sent.
RECV_EXCEPTION	0x34	Message received when a session-related exception occurs on the Node. The data field contains the name of

MessageType	ID	Description
		the session and a description of the exception.
RECV_RESET_MSG	0x35	Message received when a session gets reset. The data field contains the session that was reset.
RECV_CONNECTION_STATUS	0x36	Message received when the status session of a FIX session changes. The data field contains the session and new state of the session.
SEND_ACK	0x37	Message acknowledgment sent from the client to the server sent when the corresponding <code>RECV_MESSAGE</code> was received. This message should contain the same <code>MessageId</code> value as the <code>RECV_MESSAGE</code> to indicate which message is being acked.
RECV_ACK_MSG	0x38	Received when an outbound message has been successfully sent over a FIX session. This message contains the complete FIX message, including the header and trailer.
SESSION_STATUS_REQUEST	0x46	Requests the connection status of all FIX sessions. The Node will respond with one or more <code>RECV_CONNECTION_STATUS</code> messages.

### 7.1.3. Configuration

The example illustrates a `SocketAdapter` configured on the Source and Listener message processor chains of a FIX session listening for socket clients on port 7000.

```
<Sessions>
  <Session fixversion="4.2" heartbeat="60" counterpartycompid="BROKER" compid="CLIENT" >
    <Connections>
      <SocketConnection hostname="192.168.100.100" port="2000"/>
    </Connections>
    <SessionManager>
      <SourceMessageProcessors>
        <SocketAdapter id="socket_adapter" port="7000"/>
      </SourceMessageProcessors>
    </SessionManager>
  </Session>
</Sessions>
```

```

    </SourceMessageProcessors>
    <ListenerMessageProcessors>
        <MessageProcessorReference id="ref_socket_adapter" refid="socket_adapter"/>
    </ListenerMessageProcessors>
</SessionManager>
</Session>
</Sessions>

```

For a complete list of attributes, see the [SocketAdapter Configuration Reference](#).

## 7.1.4. Sample Client Code

The following code snippets are taken from the sample Java client located in */templates/src/com/camerontec/catalys/server/adapter/socket/test/SocketClient.java*. The snippets show the key interactions between the Java client and Socket Adapter configured on a Node.

### Class Definitions

```

//Client extends Thread to implement run()
public class SocketClient extends Thread {
    ...
    //To conveniently write to the log
    private static final ILogger logger_ = LogManager.getLogger(SocketClient.class);

    //Objects to connect to the Socket Adapter
    private InputStream inputStream_;
    private OutputStream outputStream_;
    private Socket socket_;
    ...

    //table to hold acks
    private Map<String, SocketMessage> ackTable_;
    //Transforms IFIXMessage objects to BytesSegment objects
    private IMessageTransformer transformer_;
    //Catalys FIX message object
    private IFIXMessage message_;
    ...
}

```

### Connecting to the Socket Adapter

```

socket_ = new Socket(hostname_, port_);
outStream_ = new BufferedOutputStream(socket_.getOutputStream());
inputStream_ = new BufferedInputStream(socket_.getInputStream());

```

### Reading and Processing Messages from the Node

```

try {

```

```
//Read message
SocketMessage msg = SocketMessage.readMessage(inputStream_);
//Check the MessageType and process each message type accordingly
switch (msg.type_) {
    ...
}
//Catch exceptions
} catch (Exception e) {
    logger_.error("Error processing messages. " + e);
    ...
}
```

### Acknowledge RECV\_MESSAGE if Enabled

```
//Check ack flag
if (ack_) {
    //Create SEND_ACK message from received message
    msg.type_ = SocketMessage.SEND_ACK;
    msg.writeMessage(outStream_);
}
```

### Update Acknowledge Table on the Client and Notify Any Waiting Threads

```
//Check ack flag
if (ack_) {
    synchronized (ackTable_) {
        ackTable_.put(msg.id_, msg);
        ackTable_.notifyAll();
    }
}
```

### Create and Populate an Outbound Message

```
//Instantiate message object
IFIXMessage message = new FIXMessageAsIndexedByteArray();
//Set fields on the message
message.setValue(Field.MsgType.tag, MsgType.NewOrderSingle.value);
message.setValue(Field.OrderQty.tag, 1000);
```

### Transform and Send Message

```
//Transform and cast message to a BytesSegment
BytesSegment bytesSegment = (BytesSegment)transformer_.toObject(msg, null);
//Create a SocketMessage and add the unique message id
SocketMessage sm = new SocketMessage(SocketMessage.SEND_MESSAGE,
    bytesSegment, "id:" + orderIndex_);
```

```
//Send the message
sm.sendMessage(outStream_);
```

## 7.1.5. Message Transformers

### IMessageTransformer

The [IMessageTransformer](#) interface is designed to accommodate a broad range of message formats and convert them into an internal format ([IFIXMessage](#)) that is used throughout the Catalys Node. All message data fields are transformed by an [IMessageTransformer](#) implementation. An example is the [StandardFixTransformer](#). It takes raw FIX messages in the form of byte arrays and converts them into [IFIXMessage](#) objects.

### IMessageProcessorEventTransformer

The [IMessageProcessorEventTransformer](#) interface is designed to accommodate a broad range of event formats and convert them from internal formats into messages usable by various clients. All event data fields are transformed by an [IMessageProcessorEventTransformer](#) implementation.

By default, the socket adapter uses [SocketAdapterRawTransformer](#). It converts session connection status, reset and exception events into easily parsed messages. As an example is the format for a session connection status event: `SESSION_ID:Bigbroker/CONNECTED:true`. This will appear in the data field of a `RECV_CONNECTION_STATUS` message.

## 7.1.6. Multi-Session Support

The socket adapter supports one client sending and receiving messages from multiple FIX sessions. The following example shows how to configure this:

```
<Sessions>
  <Session counterpartycompid="MARKET1" compid="BROKER" fixversion="4.2" heartbeat="60">
    <SessionManager>
      <SourceMessageProcessors>
        <SocketAdapter id="sa1" multiSession="true" port="7000"/>
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <MessageProcessorReference id="listen_sa1" refid="sa1"/>
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
  <Session counterpartycompid="MARKET2" compid="BROKER" fixversion="4.2" heartbeat="60">
    <SessionManager>
      <SourceMessageProcessors>
        <SocketAdapter id="sa2" multiSession="true" port="7000"/>
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <MessageProcessorReference id="listen_sa2" refid="sa2"/>
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
</Sessions>
```

```

</SessionManager>
</Session>
</Sessions>

```



### Note

Multi-session requires each session to have a `SocketAdapter` configured with the `multiSession` attribute set to `true` and all instances listening on the same port.

When multi-session support is enabled, the Java client is required to set the value of the `TargetCompID` field(s) to the same values configured on one of the sessions. In order for the client to send an outbound message to the `MARKET2` session in the above example, it would set the `TargetCompID` field on the message object like this:

```
message.setValue(Field.TargetCompID.tag, "MARKET2");
```

## 7.1.7. Message Acknowledgements

The socket adapter supports message ack'ing between the client and server. The socket adapter will always respond to a `SEND_MESSAGE` sent from the client with a `RECV_ACK` or `RECV_NACK`. The client can decide how to handle these message types.

If the `ackTimeout` attribute on the socket adapter is greater than 0, then the client is required to respond to `RECV_MESSAGE` messages with a `SEND_ACK` before the `ackTimeout` value (in milliseconds). The socket adapter will disconnect the client if the `SEND_ACK` is not sent before the timeout.

## 7.2. Advanced Socket Adapter

### 7.2.1. Simple vs. Advanced

The Node provides two types of socket-based adapters: the [Socket Adapter](#) and Advanced Socket Adapter (ASA). The difference between the two is shown in the following table.

Feature	Simple	Advanced
Number of Socket Clients	1	Many
Client/Server Message Acknowledgements	Yes	Yes
Inbound Message Recovery (from FIX)	No	Yes

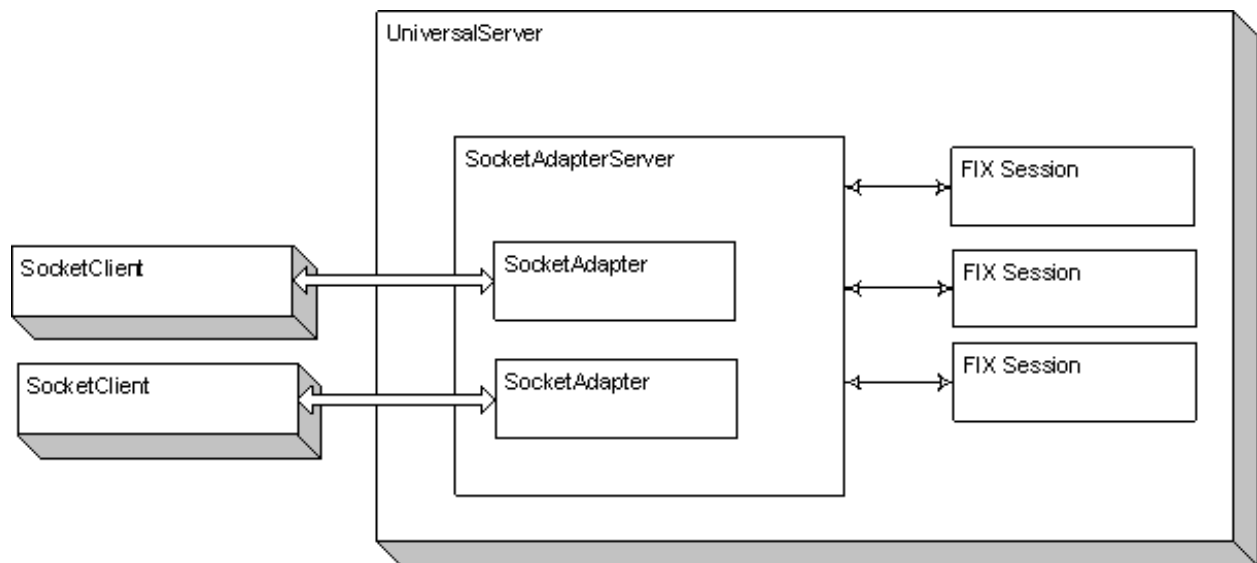


Feature	Simple	Advanced
Outbound Message Recovery (to FIX)	No	Yes
Node Command-line support	No	Yes
Number of FIX Sessions	Many	Many
Message Filtering By Session	No	Yes

In general, the Socket Adapter offers a fast and simple interface, whereas the Advanced Socket Adapter allows for greater flexibility and message recovery for both inbound and outbound messages.

## 7.2.2. Application Architecture

The architecture of a Node running the Advanced Socket Adapter looks like this:



## 7.2.3. Protocol

### 7.2.3.1. Message Format

Messages exchanged between the ASA and clients are structured as follows:

Name	Size in Bytes	Description
STX 0x02	1	Indicator for the start of the message.
MessageLength	6	The total length of the entire message excluding the <code>MessageLength</code> field itself. This field is

Name	Size in Bytes	Description
		represented as a 6-character text string padded with zeros, e.g. "000128".
MessageType	1	An 8-bit number that identifies the type/contents of the message.
AckId	Variable	A unique identifier which indicates the message must be acknowledged with the same identifier.
ETX 0x03	1	Because the AckId can be any length this byte indicates the end of the AckId field.
Data	Variable	The data byte array. The type and contents is determined by the MessageType field
EOT 0x04	1	This indicates the end of the message and is used to ensure the message has been received intact.

### 7.2.3.2. Message Data

Many of the messages use a [TableSet](#) in the data field. This is a table used to transmit array data. For example, consider the results of executing a CLI command, which are displayed on the command line like this:

```
Primary
-----
sell1

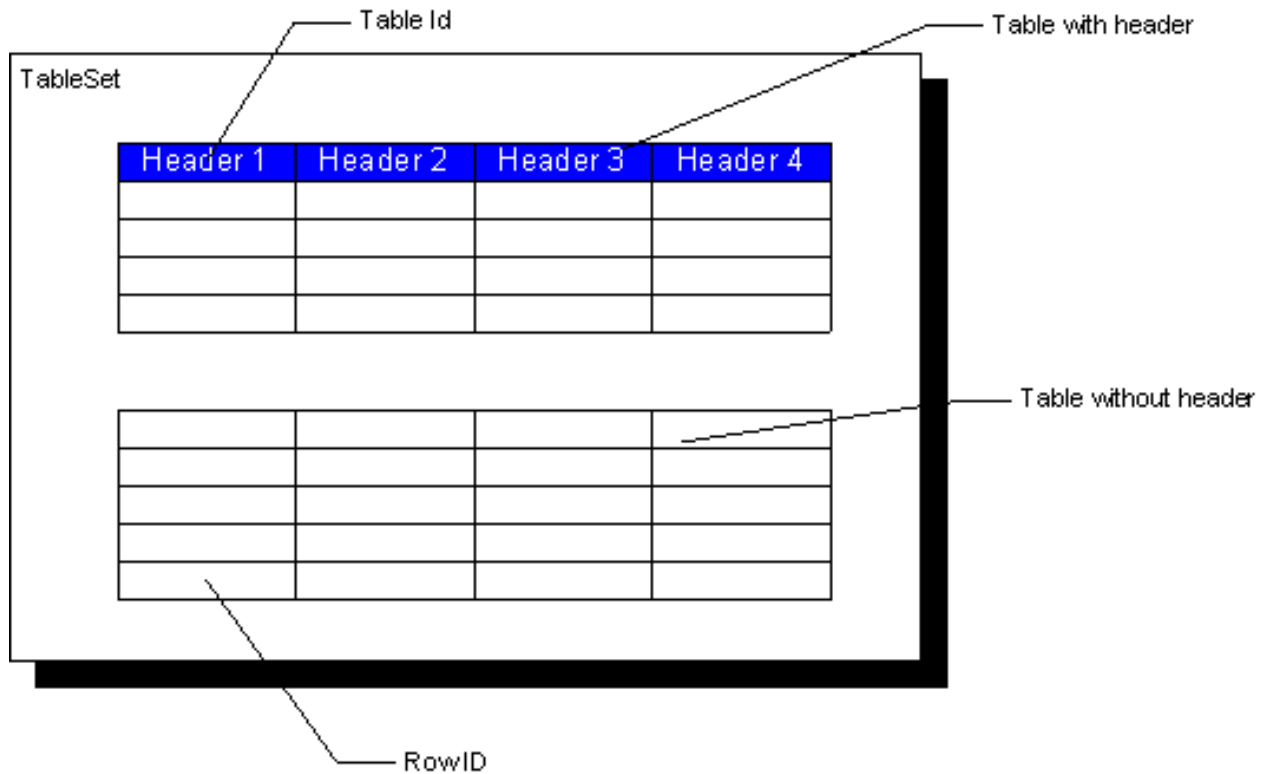
Server  Local State  Remote State Synced Active Address   Port Sent Recv
-----
sell1   LOCAL          LOCAL          true  true  127.0.0.1 9001 0   0
sell2   Disconnected   Disconnected   false false  127.0.0.1 9002 -1  0
```

When encoded as a TableSet, the data would appear as follows (with line breaks inserted for readability):

```
<SOH>Primary<ETX>
<ACK>sell1<ETX>
<SOH>Server<ETX>Local State<ETX>Remote State<ETX> \
    Synced<ETX>Active<ETX>Address<ETX>Port<ETX> \
    Sent<ETX>Recv<ETX>
<ACK>sell1<ETX>LOCAL<ETX>LOCAL<ETX>true<ETX> \
    true<ETX>127.0.0.1<ETX>9001<ETX>0<ETX>0<ETX>
<ACK>sell2<ETX>Disconnected<ETX>Disconnected<ETX> \
    false<ETX>>false<ETX>127.0.0.1<ETX>9002<ETX> \
```

```
-1<ETX>0<ETX>
```

A `TableSet` structure has more flexibility than a regular array of data, as shown in the following diagram:



Each `Table` consists of an optional header row (marked by `SOH`), and a number of data rows. Data rows can be valid (marked by an `ACK`) or invalid (marked by a `NACK`). Each field in each row (header or data) is terminated by the `ETX` character.

The valid/invalid row markers are used where the data of each row is associated with a status. For example, if the `session_disconnect all` command is issued via the `COMMAND` message, the results contained in the resultant `COMMAND_ACK` message may be different for each session. Sessions that have been successfully disconnected have an `ACK` row type, and sessions that have failed to disconnect have a `NACK` row type.

The following table summarises this encoding:

Name	Hex Value	Description
Start Of Heading	0x01	SOH Indicates a header row
Start Of ACK row	0x06	ACK Indicates an ACK row
Start Of NACK row	0x15	NACK Indicates a NACK Row

Name	Hex Value	Description
End Of Text	0x03	ETX Indicates a end of a text column

### 7.2.3.3. Message Types

The protocol uses the following message types:

MessageType	ID	Description
PROTOCOL_MSG	0x30	An inbound or outbound FIX or other protocol message.
ACK	0x32	Acknowledgement of a PROTOCOL_MSG
NACK	0x33	This is only returned by the Server and is usually due to a connection being down or the message is poorly formatted.
SESSION_EXCEPTION	0x34	Received when an exception occurs on the Node. The data field contains a text description of the exception.
SESSION_RESET	0x35	The sequence numbers have been reset on the current session.
SESSION_STATUS	0x36	Received when the session is connected or disconnected.
CONNECT	0x38	Sent by the client to connect to the Node.
CONNECT_SESSION_FILTER	0x46	Sent by the client to connect to the Node. Enables filtering by session messages received by client.
CONNECT_ACK	0x41	Returned by the Node to acknowledge a client connection.
COMMAND	0x39	Issue a command to the console command line.
COMMAND_ACK	0x40	The result of a COMMAND message.

MessageType	ID	Description
GET_MESSAGES_FROM	0x42	Resend request sent by the Socket Client.
GET_MESSAGES_FROM_ACK	0x43	Acknowledgement of GET_MESSAGES_FROM.

## CONNECT Message

This message is used to connect to the ASA. It determines the state of the connection and the state of the last connection for each session. Because there can be multiple clients connecting to a single Socket Adapter Server, each socket client must issue a connect message with a unique socket client ID in the ACK field. This unique ID will be returned in the CONNECT\_ACK message from the server.



### Note

A `CONNECT` or `CONNECT_SESSION_FILTER` message must be sent by the client within 5 seconds of the socket being established or the server will automatically disconnect.

Name	Value	Description
STX	0x02	
MessageLength		
MessageType	0x38	Indicates a <code>CONNECT</code> message
AckId	<ConnectionID>	Uniquely identifies the connection to the Node. Because multiple socket connections can be established, this ID is used to track each connection. For example: <code>Order#1234,12</code>
ETX	0x03	
Data	Variable	A <code>TableSet</code> of connections and their state.
EOT	0x04	

In the data field, connections are identified by the two parties of the connection and the state is the next message sequence number expected. For example:

## Technology Adapters

Session	SeqNumber
SellSide1/BuySide1	12345
SellSide2.subid2/BuySide2	0
SellSide6/BuySide6.BuySubid6.BuyLocationid6	345

The data field can also be used to specify a filter for the connection, by adding two extra columns to the connections: `FilterTag` and `FilterValue`. When an incoming message arrives, it will only be sent to the adapter if the tag is present and the value matches the passed in value. For example:

Session	SeqNumber	FilterTag	FilterValue
Bigbroker/Fats Fast Market 1	49		Bigbroker

If a `CompID`, `SubID` or `LocationID` contains a period, it must be escaped with the backslash character. This is necessary in order to distinguish the period from a separator between `CompID.SubID.LocationID`. For example:

Session	SeqNumber
CompID\ .With\ .Periods.SubID.LocationID/OtherSide	0
CompID.SubID\ .With\ .Periods.LocationID/OtherSide	0
CompID.SubID.LocationID\ .With\ .Periods/OtherSide	0

### CONNECT\_SESSION\_FILTER Message

This message is very similar to `CONNECT` message and may be used instead. The difference is that it enables client to choose sessions from which they want to receive messages and session events and it doesn't support single tag filter.



#### Note

A `CONNECT_SESSION_FILTER` or `CONNECT` message must be sent by the client within 5 seconds of the socket being established or the server will automatically disconnect.

Name	Value	Description
STX	0x02	
MessageLength		
MessageType	0x46	Indicates a <code>CONNECT_SESSION_FILTER</code> message

Name	Value	Description
AckId	<ConnectionID>	Uniquely identifies the connection to the Node. Because multiple socket connections can be established, this ID is used to track each connection. For example: Order#1234,12
ETX	0x03	
Data	Variable	A <code>TableSet</code> of connections and their state. Note that only messages received by server for specified connections will be delivered to the client. Connections are identified by the two parties of the connection and the state is the next message sequence number expected.
EOT	0x04	

Example CONNECT\_SESSION\_FILTER data:

```

Session                               SeqNumber
SellSide1/BuySide1                    12345
SellSide2.subid2/BuySide2              0
SellSide6/BuySide6.BuySubid6.BuyLocationid6 345

```

Like the CONNECT message, if a CompID, SubID or LocationID contains a period, it must be escaped with a backslash.

## CONNECT\_ACK Message

This is the response to the CONNECT and CONNECT\_SESSION\_FILTER message.

Name	Value	Description
STX	0x02	
MessageLength		
MessageType	0x41	Indicates a CONNECT_ACK Message
AckId	<ConnectionID>	This is the same value that was given in the CONNECT or CONNECT_SESSION_FILTER message.
ETX	0x03	

Name	Value	Description
Data	Variable	A <code>TableSet</code> of: the <code>AckId</code> of the last outbound message processed by the Node, and a list of sessions with their last reset time (in milliseconds, as returned by <code>System.currentTimeMillis()</code> ). If the last reset time is unknown then this field will contain the value of -1.
EOT	0x04	

Example CONNECT\_ACK data:

```
AckId
Order#1234,12
Session          LastResetTime
SellSide1/BuySide1 1276824689515
SellSide2/BuySide2 -1
```

## PROTOCOL Message

The Socket Client sends and receives FIX messages from the Node as:

Name	Value	Description
STX	0x02	
MessageLength		
MessageType	0x30	PROTOCOL_MSG
AckId	<Identifier>	Uniquely identifies the session and the sequence number of the message. The socket clients must generate their own unique IDs. Protocol messages without this ID will not be acknowledged. The unique identifier used by the Node uses the parties to the session and the sequence number of the message in the form <code>TargetComp.Sub.Loc/SenderComp.Sub.Loc:SeqNum</code> . For example: <div> <pre>Sell.sub/buy:514 Target/sender.senderSub.Location:7365</pre> </div>



Name	Value	Description
ETX	0x03	
Data	Variable	The raw FIX message
EOT	0x04	

### PROTOCOL\_ACK Message

This is sent by the client or the server to acknowledge a PROTOCOL message.

Name	Value	Description
STX	0x02	
MessageLength		
MessageType	0x32	ACK
AckId	<Identifier>	Uniquely identifies the connection and the sequence number of the message. Refer to the description of the AckId field of the <a href="#">PROTOCOL</a> message.
ETX	0x03	
Data	Variable	No data (except from the server when <a href="#">addDataToAckMsg</a> attribute is specified).
EOT	0x04	

### COMMAND\_MSG Message

[CLI](#) commands are issued via this message. It is the only message that can be processed by the ASA before the connect sequence is completed.

Name	Value	Description
STX	0x02	
MessageLength		
MessageType	0x39	COMMAND_MSG

Name	Value	Description
AckId	<Identifier>	A unique ID. The resulting COMMAND_ACK will have the same AckId value.
ETX	0x03	
Data	Variable	Command string, e.g. s_stats all
EOT	0x04	

## COMMAND\_ACK Message

The Socket Adapter acks COMMAND messages with a COMMAND\_ACK message.

Name	Value	Description
STX	0x02	
MessageLength		
MessageType	0x40	COMMAND_ACK
AckId	<Identifier>	The AckId value of the original COMMAND_MSG message.
ETX	0x03	
Data	Variable	TableSet of results of the command
EOT	0x04	

Example COMMAND\_ACK data:

#	TargetCompId	SenderCompId	Status	Blocked	Locked	Active	InSeqNo	OutSeqNo	PendingIn
15	Bigbroker	Fats Fast Market	UP	N	N	Y	201	201	0
1	Bigbroker1	Fats Fast Market1	DOWN	N	N	Y	-1	-1	0

## GET\_MESSAGES\_FROM Message

This message asks the Socket Adapter to send messages beginning at a specific sequence number. Messages already in transit to the client application that are awaiting acknowledgement will be reset (no acknowledgment required) and the conversation will continue with the sequence number given (or the next sequence greater than the given number). For example, if the next requested sequence number is

100, but the session started the day at 200 (because of manual intervention), messages from 200 and up will be sent.

Note that Logon and Logout messages will not be retrieved by this command; they are sent directly to the client without being stored on the server.

Name	Value	Description
STX	0x02	
MessageLength		
MessageType	0x42	Indicates a GET_MESSAGES_FROM message
AckId	<ConnectionID>	This value must be the same value that was supplied in the original CONNECT message.
ETX	0x03	
Data	Variable	A TableSet of sessions and the next sequence number requested. For example: <div> <pre> Session          SeqNumber SellSide1/BuySide1 123 </pre> </div>
EOT	0x04	

## GET\_MESSAGES\_FROM\_ACK Message

This message is sent in response to the GET\_MESSAGES\_FROM message.

Name	Value	Description
STX	0x02	
MessageLength		
MessageType	0x43	Indicates a GET_MESSAGES_FROM_ACK message
AckId	<ConnectionID>	This value is the same as that supplied in the GET_MESSAGES_FROM message.
ETX	0x03	

Name	Value	Description
Data	Variable	A <code>TableSet</code> containing the last <code>AckId</code> processed by the Socket Adapter. <div> <pre>AckId Order#1234,12</pre> </div>
EOT	0x04	

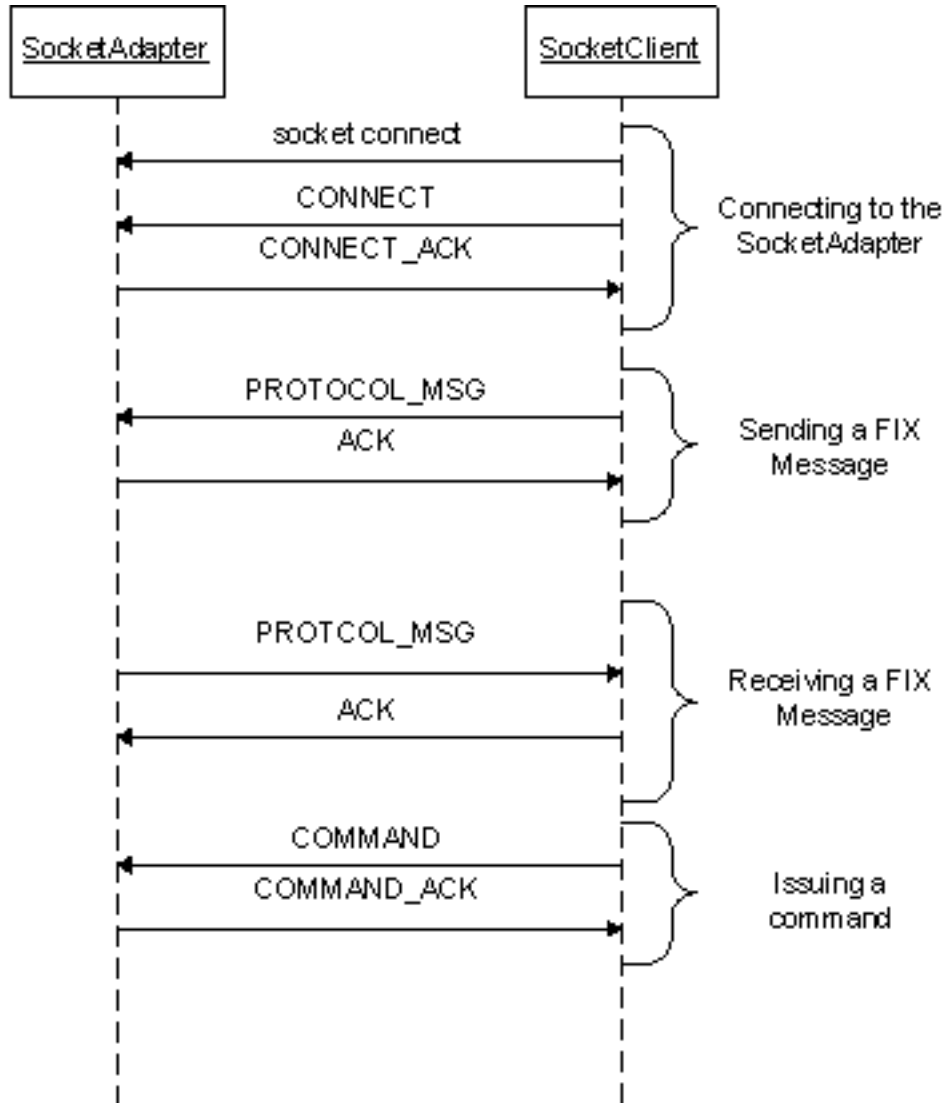
## Session Event Messages

Session event messages are sent to indicate changes in a session's state. Because of the fleeting nature of these events they are not ACKed nor can they be recovered. They are sent immediately and there is no guarantee of order with protocol messages. For example, if a FIX counter-party disconnects, a disconnect event could be received before the last few remaining messages are processed. The following table shows the format of session events messages.

Name	Value	Description
STX	0x02	
MessageLength		
MessageType	0x34, 0x35, 0x36	SESSION_EXCEPTION, SESSION_RESET, SESSION_STATUS
AckId	<Identifier>	This contains the "Full Name Identification" of the session. It must not be ACKed.
ETX	0x03	
Data	Variable	<ul style="list-style-type: none"> <li>SESSION_EXCEPTION: A text description of the exception.</li> <li>SESSION_STATUS: "CONNECT" or "DISCONNECT"</li> <li>SESSION_RESET: Reset time in milliseconds</li> </ul>
EOT	0x04	

## Message Sequence

The following diagram shows the messages that are exchanged over the course of a socket connection.



### 7.2.3.4. Other Protocol Notes

#### Session Reset

When a session is "reset" its inbound and outbound sequence number are reset to 1. The Socket Adapter will send the client a SESSION\_RESET message and will reset the internal client inbound sequence numbers back to 1.

#### Logon and Logout FIX Messages

Logon and logout messages received by the ASA from a FIX counter party are delivered directly to the client. They do not get stored and hence they are not recoverable. This means the GET\_MESSAGES\_FROM message will not resend logon or logout messages. To determine the current state of a session, a COMMAND message can be issued from the client to the Node.

## Filtering

A Socket Adapter connection can be configured to filter message based on a tag and value. These are supplied in the data field of the CONNECT message.

A Socket Adapter connection can also be configured to filter messages and session events based on session. Session details are supplied in the data field of the CONNECT\_SESSION\_FILTER message.

## 7.2.4. Configuration

The ASA is a message processor that may be configured on a session. Refer to the [configuration reference](#) for a complete list of its attributes. Since multiple clients can connect to a single Socket Adapter Server, all clients connecting on the same port must have identical attributes. However, the only attributes excluded from this condition that may be different are `id`, `msgTypes`, and `rejectIfNotConnected`.

## Configuration Example

The following shows an example configuration. Here, three sessions are each configured with an `AdvancedSocketAdapter` message processor. Two of them listen on the same port (7200) while the third listens on another port (7201).

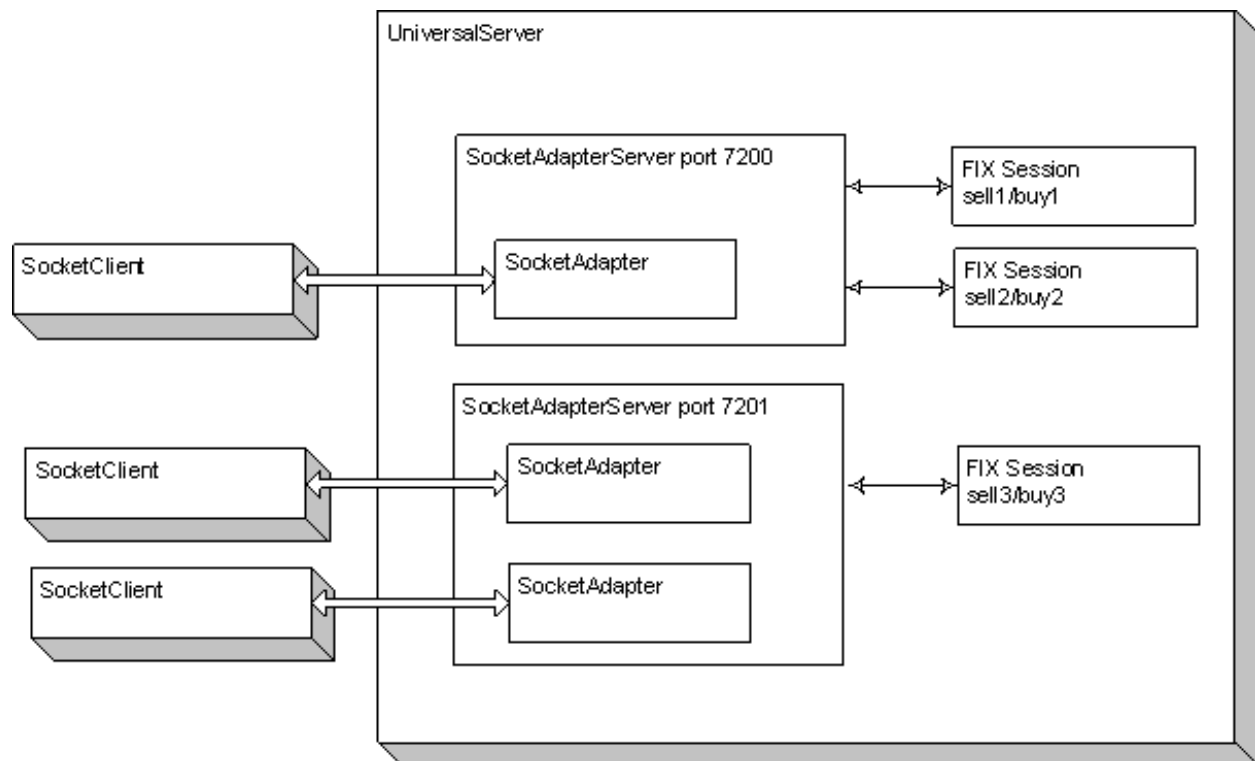
```
<Config>
  <Application id="SellSide" unqueuedMode="true">
    <Sessions>
      <Session counterpartycompid="buy1" compid="sell1" fixversion="4.2" heartbeat="60">
        <Connections>
          <SocketConnection id="sock_1" listenport="2000"/>
        </Connections>
        <SessionManager>
          <SourceMessageProcessors>
            <AdvancedSocketAdapter port="7200" ackTimeout="0" id="asa1"/>
          </SourceMessageProcessors>
          <ListenerMessageProcessors>
            <MessageProcessorReference id="asa2" refid="asa1"/>
          </ListenerMessageProcessors>
        </SessionManager>
      </Session>

      <Session counterpartycompid="buy2" compid="sell2" fixversion="4.2" heartbeat="60">
        <Connections>
          <SocketConnection id="sock_2" listenport="2000"/>
        </Connections>
        <SessionManager>
          <SourceMessageProcessors>
            <AdvancedSocketAdapter port="7200" ackTimeout="0" id="asa3"/>
          </SourceMessageProcessors>
          <ListenerMessageProcessors>
            <MessageProcessorReference id="asa4" refid="asa3"/>
          </ListenerMessageProcessors>
        </SessionManager>
      </Session>
    </Sessions>
  </Application>
</Config>
```

## Technology Adapters

```
<Session counterpartycompid="buy3" compid="sell3" fixversion="4.2" heartbeat="60">
  <Connections>
    <SocketConnection id="sock_3" listenport="2000"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors>
      <AdvancedSocketAdapter port="7201" ackTimeout="0" id="asa5"/>
    </SourceMessageProcessors>
    <ListenerMessageProcessors>
      <MessageProcessorReference id="asa6" refid="asa5"/>
    </ListenerMessageProcessors>
  </SessionManager>
</Session>
</Sessions>
</Application>
</Config>
```

Here is the corresponding application architecture:



For a working demonstration of the ASA, see the template in *templates/SocketAdapter/Advanced*. The source code for the `SocketClient` used in the template can be found in *templates/src/com/camerontec/catalys/server/adapter/socket/advanced/test*.

### 7.2.5. Command-Line Operations

The ASA provides a [CLI](#) command, `asa_list`, that can be used to show a list of connected ASA clients and the session filter list for each client.

## 7.3. Embedded Catalys Node Adapter

### 7.3.1. Introduction

The Embedded Catalys Node Adapter is designed to work with another Java application which starts up the Node and treats it as a Java object running in the same process. The application is able to send and receive FIX messages across multiple sessions and all the features of the Node (HA, Rules Engine, XML configuration, etc.) are still supported.

### 7.3.2. Adapter Operation

Normally custom code is implemented as a [Custom Message Processor](#) configured on a session. The Embedded Catalys Node Adapter is designed to provide the same internal event and FIX message sending and receipt capabilities from within a standalone application by embedding a Node.

Specifically, the following features are supported:

- Receipt of incoming FIX messages
- Sending of outgoing FIX messages
- FIX session connection status events
- FIX session exception events
- FIX session reset events

In order to use the Embedded Catalys Node Adapter it is necessary to:

- Configure a Node with one or more `EmbeddedCatalysNodeAdapter` elements.
- Create a Java application which instantiates and starts a single [EmbeddedCatalysNodeAdapter](#) instance, supplying this instance with:
  - A set of [application parameters](#) compatible with those of the `CatalysServer` class that will be created by this instance, including the required `-xmlconfig` which references the XML configuration file.
  - For each `EmbeddedCatalysNodeAdapter` element in the configuration, a map of its ID string to a corresponding [IFIXListenerCallback](#) component, whose methods will be called in response to received FIX session events and messages.
- Implement the facility for sending FIX message from the Java application by using either [EmbeddedCatalysNodeAdapter.send](#) or the [IFIXSourceProxy](#) interface.

### 7.3.3. Configuration

The following example shows how to configure an `EmbeddedCatalysNodeAdapter` on both the Source and Listener side of a session:



```

<Sessions>
  <Session counterpartycompid="BROKER" compid="MARKET" fixversion="4.2" heartbeat="60">
    <Connections>
      <SocketConnection id="scl" hostname="localhost" port="2000"/>
    </Connections>
    <SessionManager>
      <SourceMessageProcessors>
        <EmbeddedCatalysNodeAdapter id="ecna1"/>
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <MessageProcessorReference id="ecna1_listen" refid="ecna1"/>
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
</Sessions>

```

For a complete list of attributes, see the [EmbeddedCatalysNodeAdapter Configuration Reference](#).

For details on how to add custom JMX instrumentation to your adapter, refer to the [Programming Guide](#).

## 7.3.4. Templates and Example Client Code

Example configurations and source code for clients for the Embedded Adapter are packaged along with the Node. Different templates exist to demonstrate different aspects of the adapter.

- Templates located in *templates/EmbeddedCatalysNodeAdapter/*
- Source code located in *templates/src/com/camerontec/catalys/server/adapter/embedded/client/*

A ReadMe.txt is located in each template directory providing full details on what the template does and how each is run.

### Unqueued Template

This template shows the most standard usage of the adapter where inbound messages come to the `onMessageFromFIX(...)` event handler one at a time. The messages are never queued in the Node or adapter.

### Queued Template

This template shows an alternative way to process inbound messages. When messages come to the `onMessageFromFIX(...)` event handler they are placed in a `BlockingQueue<?>` data object for consumption sometime in the future. This can be useful if the Java client using the embedded adapter is a slow consumer or processor of messages.

## MultiSession Template

This template shows how the embedded adapter can support more than one session. The key is to configure the `EmbeddedCatalysNodeAdapter` element like you would with one session and then refer to it using `MessageProcessorReference` elements on the rest of the sessions.

When multiple sessions need to be supported, the embedded client is required to set the value of the `TargetCompID` field(s) to the same values configured on one of the sessions. In order for the client to send an outbound message to the `BROKER3` session for instance, it would set the `TargetCompID` field on the message object like this:

```
message.setValue(Field.TargetCompID.tag, "BROKER3");
```

## UnqueuedAddSession Template

This template is the same as the `Unqueued` template but demonstrates how a session can be added on the fly. The Node is started without any sessions configured, then during runtime, the XML config file is updated and the `config_reload` command is issued which detects and adds the session at runtime.

## Embedded Sender Class

This example class shows how the client can send outbound messages down a configured session by creating a message object, populating fields and then calling:

```
IFIXProxy.fireMessage(IFIXMessage);
```

Relevant JavaDocs are located here:

- [EmbeddedCatalysNodeAdapterUnqueuedListener](#)
- [EmbeddedCatalysNodeAdapterQueuedListener](#)
- [EmbeddedCatalysNodeAdapterSender](#)



### Note

There are many other possibilities when implementing your Java client to interact with the Embedded Adapter and the code we provide is for demonstration purposes only.

## 7.3.5. Performance

There are 2 primary considerations when processing inbound events and FIX messages:

1. Threading model
2. Processing time

The Embedded Adapter delivers events and inbound FIX messages to the embedding application via one or more [IFIXListenerCallback](#) instances. Each `IFIXListenerCallback` instance is registered with one or more [EmbeddedCatalysNodeAdapterProcessor](#) (the class corresponding to the `EmbeddedCatalysNodeAdapter` message processor) instances.

The main point to note is that all events and inbound FIX messages are delivered in the same thread as to the underlying `EmbeddedCatalysNodeAdapterProcessor` itself - no buffering is performed. Since the Node is multi-threaded, typically with a separate delivery thread per FIX session, then listeners registered against multiple FIX sessions are likely to receive events and messages concurrently from separate threads. This is not the case for listeners registered with a single FIX session, unless messages are being routed between FIX sessions inside the Node.

Since events and messages are delivered by internal Node threads, a slow running or blocking listener is likely to directly affect the operation of the server itself. Therefore it is recommended never to block on receipt of inbound events or FIX messages. If this is not possible to guarantee then it is recommended to queue the incoming events and messages and process them in a separate thread owned by the embedding application.

We call the mode in which events and messages are processed directly **unqueued** mode and the mode in which events and messages are queued (or otherwise buffered) for subsequent processing in another thread **queued** mode.

## 7.4. JMS Adapter

### 7.4.1. Introduction

The JMS adapter enables the Node to work with any JMS compliant messaging product (such as FioranoMQ, JBoss, SonicMQ, SwiftMQ, Websphere MQ, Sonic MQ, Oracle WebLogic and OpenJMS). Together with the Node, FIX messages can be sent and received to and from any application connected over JMS.

### 7.4.2. JMS Provider Installation

The JMS provider's software must be installed according to their instructions. These instructions will normally involve additional jar files to the classpath. Typically, the provider's JMS server will need to be started, and the required JMS queues and topics added.

By default, the JMS adapter uses Java's standard naming interface (JNDI) to link up with a particular provider's implementation of JMS. With the Node JMS adapter, the JNDI information is specified in a properties file called *jndi.properties*. This directory where this file resides should be on the classpath.

For example, if the current directory (".") is in the classpath, then the *jndi.properties* file can be located in the same directory where the Node is started.

### 7.4.3. Configuration

There are two JMS message processors:

- **JmsProducer:** Listens for inbound FIX messages received on a session and writes to JMS queue. For a complete list of attributes, see the [JmsProducer Configuration Reference](#).
- **JmsConsumer:** Reads messages from a JMS queue and sends them out a session. For a complete list of attributes, see the [JmsConsumer Configuration Reference](#).

The configuration example below shows a JMS adapter configured on a session. The `JmsConsumer` takes messages off a queue or topic and sends them out a FIX session. The `JmsProducer` takes inbound FIX messages and places them on a queue or topic. The adapter in the example is using `primaryQCF` as the name of the queue connection factory. This is the default factory name provided by FioranoMQ.

```
<Sessions>
  <Session counterpartycompid="EXCHANGEA" compid="FIRM1" fixversion="4.2" heartbeat="60">
    <Connections>
      <SocketConnection hostname="192.168.10.100" port="2000" />
    </Connections>
    <SessionManager>
      <SourceMessageProcessors>
        <JmsConsumer type="queue" destination="EXCH_OUTBOUND" factory="primaryQCF" />
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <JmsProducer type="queue" destination="BUY_INBOUND" factory="primaryQCF" />
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
</Sessions>
```

By default, the `JmsProducer` uses `DefaultJMSTransformer` as the message transformer, which puts JMS messages of type `javax.jms.ObjectMessage` on the queue. To place JMS messages of other types (bytes, object, text) on the queue, an additional `Properties` element must be configured to set the `object.format` attribute of the transformer. The following formats are supported:

- bytes: Put JMS message of type `javax.jms.BytesMessage` on the queue
- object: Put JMS message of type `javax.jms.ObjectMessage` on the queue
- text: Put JMS message of type `javax.jms.TextMessage` on the queue

In this example we set the format to `text`:

```
<JmsProducer type="queue" destination="BUY_INBOUND" factory="primaryQCF">
  <Properties id="myProducerProperties">
    <Property name="object.format" value="text" />
  </Properties>
</JmsProducer>
```

## 7.4.4. Templates

There are two templates packaged inside the Node that demonstrate the functionality of the JMS Adapter: *templates/JMSAdapter/ActiveMQ/topic* and *templates/JMSAdapter/ActiveMQ/queue*. One interacts with JMS queues and the other with JMS topics.

The *ReadMe.txt* will guide you through the templates. Some things to note:

- Place the .jar files for your JMS library in the */ext* directory of your install.
- There is a *jndi.properties* file in each template. The directory where it resides is required to be on the classpath.
- *readJMS.[sh|bat]* starts up a client application that receives JMS messages over a topic or queue. The connection factory and queue/topic names are passed as application parameters.
- *writeJMS.[sh|bat]* starts up a client application that places JMS messages on a topic or queue. The connection factory and queue/topic names are passed as application parameters.
- The source code for the classes the readJMS and writeJMS call is available in *templates/src/com/camerontec/catalys/server/adapter/jms/test/jndi/*

## 7.5. Solace Adapter

### 7.5.1. Introduction

The Solace Adapter enables the Node to send and receive FIX messages to and from Solace Systems messaging appliances.

The adapter uses the Solace Message API for Java (known as JCSMP). Versions 6.1, 6.2 and 7.1.0.207 of the JCSMP API are confirmed to be supported by the adapter.

### 7.5.2. Installation

A valid license containing the Solace Adapter feature is required.

The following JCSMP API jar files must be available on the Node's classpath. These files are not bundled with the Node and must be obtained from Solace.

- sol-jcsmp-6.x.x.x.jar
- sol-common-6.x.x.x.jar

## 7.5.3. Functionality

### Delivery Modes

The adapter has two delivery modes:

- **Guaranteed:** Provides "once and only once" delivery. When publishing messages to an appliance, the adapter expects delivery acknowledgements. When consuming from the appliance, the adapter acknowledges each message. This is the default mode.
- **Direct:** Provides "at most once" delivery. No acknowledgements are sent or received, and successful delivery is not guaranteed. Direct mode provides higher message throughput compared to guaranteed mode, but with a trade-off in reliability.

### Message Transformation

The core function of the adapter is to transform Solace messages to/from Catalys [IFIXMessages](#). Two message transformers are bundled with the adapter:

- [SolaceRawTransformer](#): Transforms an `IFIXMessage` to/from a Solace `BytesMessage`. The payload is a `byte[]` representation of a raw FIX message. This transformer provides the best performance and is enabled by default.
- [SolaceStringTransformer](#): Transforms an `IFIXMessage` to/from a Solace `TextMessage`. The payload is a `String` FIX message. This transformer incurs a garbage collection penalty as it creates new `String` objects.

Users can also provide a custom message transformer if an alternative Solace message format or payload is required. The transformer must implement [ISolaceMessageTransformer](#).

## 7.5.4. Solace Adapter Components

### 7.5.4.1. SolaceMessageListener

Listens for messages from a FIX session and publishes them to a Solace destination.

#### Destinations

The Solace destination can be either a topic or a queue. When using topics, each message can be published to the same topic, or the topic can be dynamically resolved on a per-message basis. Dynamic topic resolution is delegated to a user-provided implementation of [ISolaceTopicResolver](#). An [example](#) is illustrated below.

## Error Handling

It is possible that an inbound FIX message cannot be published to Solace. For delivery failures related to the Solace session (e.g. the appliance is unreachable or the session has unexpectedly closed), the Node will reject the inbound FIX message and deliver a reject message to the counterparty.

As an alternative to firing rejects, a [ListenerMessageBuffer](#) can be configured to buffer inbound FIX messages while the Solace session is unavailable. When the Solace session resumes, the `SolaceMessageListener` will attempt to redeliver the buffered messages to the appliance.

For publishing failures which are not related to the Solace session (i.e. the appliance rejects the message for some reason), the `SolaceMessageListener` will reject the inbound FIX message. Depending on the FIX version in use, the Node will send either a `BusinessMessageReject` (4.2 and later) or a `Reject` (4.1 and older) to the counterparty.

### 7.5.4.2. SolaceMessageSource

Consumes messages from a Solace source and fires them to a FIX session.

#### Sources

When consuming in guaranteed mode, the consumer uses a Solace `FlowReceiver` to read messages asynchronously from a queue. Therefore, topics cannot be used as sources in this mode.

In direct mode, a Solace `XMLMessageConsumer` is used to read messages asynchronously from a topic. Queues are not supported as sources in direct mode.

Each FIX session can be configured with its own Solace source, e.g. one queue per FIX session. Alternatively, multiple FIX sessions can be configured to use the same Solace source. In this case, the consumed messages must contain the necessary `TargetCompID/TargetSubID/TargetLocationID` tags in order for the adapter to fire the message to the proper FIX session.

The adapter will create only one exclusive consumer per Solace source; this consumer will be re-used by each `SolaceMessageSource` configured to read from that source. A unique Solace source is identified by the combination of host/vpn/destination.

#### Message Consumption

By default, message consumption will commence once the FIX session has logged on, and consumption will stop when the FIX session logs off. This behavior can be changed via the `waitForLogon` attribute.

When multiple FIX sessions are sourced from the same queue or topic, message consumption will commence when at least one of the FIX sessions is logged on. If a message is consumed for a FIX session that is not logged on, FIX delivery will fail and the Solace message will be subject to [error handling](#).

## Error Handling

Consumed messages that cannot be transformed or fired to a FIX session are either:

- Published to an errorTopic or errorQueue.

This is the simplest and fastest error handling approach and can be used in either guaranteed or direct mode. When enabled, failed messages are copied, re-published to a designated Solace destination and then acknowledged. The error message can be enriched with metadata indicating the cause of the failure by providing a custom class which implements the [ISolaceMessageTransformer](#) interface.

- Published to the Dead Message Queue (DMQ).

The appliance moves the failed message to the DMQ when Maximum Message Redelivery (MMR) is exceeded.

This option is only supported in guaranteed mode and has the following requirements:

- The DMQ must be created in the VPN to which the adapter is connecting
- The source queue's MMR setting must be non-zero
- Messages published to the source queue must have the DMQ eligibility flag set to true

When the DMQ option is used, each message is consumed in a transaction; the transaction is committed if the message is successfully delivered to FIX, otherwise the transaction is rolled back.

After a rollback, the appliance decides what to do with the message based on the MMR value. The appliance will redeliver the message to the Solace Adapter (which will make another FIX delivery attempt) until the MMR value is reached.

Therefore it is important that the MMR value is not 0 when using the DMQ option, otherwise the appliance will infinitely attempt to redeliver the message.

Once the MMR limit is reached and the Solace Adapter still has not successfully handled the message, the appliance will move the message to the DMQ. Due to acknowledgement windowing, all other messages in the current window will also be moved to the DMQ. If this is not the desired behavior, the following must be configured:

- `SUB_ACK_WINDOW_SIZE` must be set to 1
- The "Max Delivered Unacked Msgs Per Flow" queue property must be set to 1. This obviously eliminates the performance benefit of acknowledgement windowing and will result in lower message throughput.

## Message Redelivery

The JCSMP API uses acknowledgement windowing when consuming in guaranteed mode. The API can receive and process up to 255 messages before delivering any acknowledgments back to the appliance.



If consumption is interrupted before entirely processing the current window, the unacknowledged messages remaining in the window will be redelivered from the appliance when consumption resumes. These messages are marked with a "redelivered" flag.

In outage situations where the Node unexpectedly fails (in either standalone or high-availability mode), depending on the point at which the failure occurs, it's possible that acknowledgements sent from the API do not reach the appliance. When consumption resumes (i.e. the Node is restarted or fails over to a secondary HA node), the appliance will deliver messages starting from where the last acknowledgement was received. Therefore the next window may contain messages which were already consumed, yet unacknowledged from the perspective of the appliance. This can result in duplicate messages being delivered to FIX.

To account for this situation, for all messages whose redelivered flag is true, the PossResend(97) tag will be added to the corresponding outbound FIX message with a value of true.

The windowing feature can also affect message delivery in non-outage situations, for example when message consumption is interrupted due to FIX session logoff. Any messages in the current window which were not delivered to FIX will remain unacknowledged, yet will be flagged as "redelivered" when consumption resumes. The corresponding FIX messages will also contain the PossResend tag.

To mitigate the occurrence of redelivered messages, the `SUB_ACK_WINDOW_SIZE` JCSMP property can be set to 1 so that only one acknowledgement is ever in flight. This however eliminates the performance benefit of acknowledgement windowing and will result in lower message throughput.

## 7.5.5. Configuration

### SolaceMessageListener

The following is a typical configuration for the `SolaceMessageListener`:

```
<Session counterpartycompid="CLIENT" compid="BROKER" fixversion="4.4">
  <Connections>
    <SocketConnection id="sc" listenport="2002"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors>
      <!-- Required for SolaceMessageListener to fire rejects -->
      <MessageProcessorReference id="ref" refid="solaceListener"/>
    </SourceMessageProcessors>
    <ListenerMessageProcessors>
      <SolaceMessageListener id="solaceListener"
        host="10.1.1.1:12345"
        username="user"
        vpnName="vpn"
        destinationTopic="topicName"
      />
    </ListenerMessageProcessors>
  </SessionManager>
</Session>
```

For a complete list of attributes, see the [SolaceMessageListener configuration reference](#).

## SolaceMessageSource

The following is a typical configuration for the `SolaceMessageSource`:

```
<Session counterpartycompid="EXCHANGE" compid="BROKER" fixversion="4.4">
  <Connections>
    <SocketConnection id="sc" listenport="2002"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors>
      <SolaceMessageSource id="solaceSource"
        host="10.1.1.1:12345"
        username="user"
        vpnName="vpn"
        sourceQueue="queue"
        errorTopic="errTopic"
      />
    </SourceMessageProcessors>
    <ListenerMessageProcessors/>
  </SessionManager>
</Session>
```

For a complete list of attributes, see the [SolaceMessageSource configuration reference](#).

## Templates

Fully working templates that demonstrate the Solace Adapter can be found in *templates/SolaceAdapter* of the Node installation. There are two templates:

- **Simple:** The Node acts as a broker between one client and one exchange session. On the client session, inbound messages are published to a Solace topic. On the exchange session, outbound messages are consumed from a Solace queue and fired to FIX.
- **MultiSession:** This template is similar to the Simple template, except there are multiple client and exchange sessions publishing to and consuming from multiple Solace destinations. The template demonstrates how to use a `MessageProcessingTemplate` to simplify the configuration of multiple Solace components.

## Dynamic Topic Resolution

In this example, a `SolaceMessageListener` is configured to publish messages from FIX to a dynamically resolved topic based on the Symbol(55) tag, for example "symbol/IBM".

The following custom [ISolaceTopicResolver](#) simply retrieves the symbol value from each message and returns the dynamic topic name as a `String`.

```
package com.example;
```

```
import com.camerontec.catalys.core.message.IFIXMessage;
import com.camerontec.catalys.server.adapter.solace.ISolaceTopicResolver;
import com.camerontec.catalys.util.data.MissingDataException;

public class ExampleTopicResolver implements ISolaceTopicResolver
{
    @Override
    public String resolveTopic(final IFIXMessage fixMessage,
                              final String destinationTopicAttribute)
    {
        String symbol = null;
        try
        {
            symbol = fixMessage.getValueAsString(55);
        }
        catch (MissingDataException ex) {}

        // note that destinationTopicAttribute is "symbol/%s"
        return String.format(destinationTopicAttribute, symbol);
    }
}
```

This configuration snippet illustrates how to refer to the custom class. The class must be available on the classpath of the Node.

```
<SolaceMessageListener
  id="solaceListener"
  host="10.1.1.1"
  username="user"
  vpnName="vpn"
  destinationTopic="symbol/%s"
  topicResolverClass="com.example.ExampleTopicResolver" />
```

## 7.5.6. Performance Tuning

To tune publishing and consuming performance, a properties file containing JCSMP settings can be provided to a `SolaceMessageListener` and `SolaceMessageSource`. For example:

```
jcsmp.SUB_ACK_WINDOW_SIZE=255
jcsmp.CLIENT_CHANNEL_PROPERTIES.tcpNoDelay=false
```

The path to the properties files is specified as follows:

```
<SolaceMessageListener id="solaceListener"
  host="10.1.1.1"
  username="user"
  password="pass"
  vpnName="vpn"
  destinationTopic="someTopic"
  jcsmpPropertiesFile="/path/to/jcsmp.properties" />
```

### 7.5.6.1. Optimizing For Higher Throughput

The following settings will likely yield higher message throughput, although you may see different results in your environment.

#### **SolaceMessageListener**

Guaranteed Mode:

- Transformer:

```
com.camerontec.catalys.server.adapter.solace.SolaceRawTransformer
```

- JCSMP properties:

```
jcsmp.PUB_ACK_WINDOW_SIZE=255
```

```
jcsmp.CLIENT_CHANNEL_PROPERTIES.tcpNoDelay=false
```

Direct Mode:

- Transformer:

```
com.camerontec.catalys.server.adapter.solace.SolaceRawTransformer
```

- JCSMP properties:

```
jcsmp.CLIENT_CHANNEL_PROPERTIES.tcpNoDelay=false
```

#### **SolaceMessageSource**

Guaranteed Mode:

- Transformer:

```
com.camerontec.catalys.server.adapter.solace.SolaceRawTransformer
```

- Error handling:

```
errorTopic Or errorQueue
```

- JCSMP properties:

```
jcsmp.SUB_ACK_WINDOW_SIZE=255
```

Direct Mode:

- Transformer:

```
com.camerontec.catalys.server.adapter.solace.SolaceRawTransformer
```

### 7.5.7. Using the Adapter with Catalys Node High Availability

The Solace Adapter is fully supported in high availability configurations. Because Catalys HA operates in an active/passive configuration, only one member of the cluster (the primary) will maintain an active connection to the Solace appliance at any given time.

The acknowledgement windowing feature of the JCSMP API has implications when running in HA mode. See the [Message Redelivery](#) section for more details.

## 7.5.8. JCSMP API Logging

The Solace JCSMP API uses the Jakarta Commons Logging (JCL) framework. By default, this logs to the console only and does not integrate with SLF4J, the logging framework used by the Node.

In order to produce unified logging so that JCSMP and the Node write to the same log, JCL logging statements must be "bridged" to SLF4J. To enable the bridge, *jcl-over-slf4j.jar* must be available on the classpath of the Node. This jar should be listed before *catalys-node.jar* and any other logging-related jar files. No additional configuration steps are necessary.

With the bridge in place, JCSMP API logging can be [customized](#) using *logback.xml*.

Visit the [SLF4J](#) site for more information and to download the bridge jar file.

## 7.6. IBM MQSeries Adapter

### 7.6.1. Introduction

The IBM MQ Adapter is designed to work with the Node and IBM's messaging platform. Together with the Node, FIX messages can be sent and received to and from any application connected over IBM MQSeries queues or topics.

### 7.6.2. MQSeries Installation

IBM MQSeries should be installed per the IBM documentation. The IBM MQ Adapter is a built-in message processor in the Node. It also requires the standard IBM MQSeries jar files be added to the classpath.

The adapter is compiled against the MQSeries v7.0 Java libraries. Other MQSeries/WebSphere MQ versions should work with the adapter, but are not officially supported. Note that IBM always provides a JMS bridge, so our [JMS adapter](#) will also work with all MQSeries/WebSphere MQ versions.

The adapter was originally developed for the WebSphere MQ v5.3 Java libraries and is currently compiled against **v7.0**. WebSphere MQ v7.0 queue managers and clients interpolate with queue managers and clients from previous versions of the WebSphere MQ or MQSeries products ([read more](#)). The adapter has been tested thoroughly on versions v5.3, v6.0 and v7.0 of the IBM product and we provide support on all of them, however we do recommend v7.0.

### 7.6.3. Typical Configuration and Message Conversion

A typical scenario for message interaction between the Node and a MQSeries application might be:

## Technology Adapters

1. A FIX counterparty establishes a socket connection with the Node
2. The Node writes inbound FIX messages to a queue or topic
3. The customer application reads inbound messages from the queue or topic
4. The customer application writes outbound messages to another queue or topic
5. The Node consumes messages from the queue or topic
6. The Node sends these messages out a FIX session to a counterparty

The MQ Adapter does not assume that the customer application is written in Java. It can be written in any programming language that can communicate with the IBM WebSphere technology. MQSeries supports a very simple message format and makes few assumptions about the message payload. The body of an MQSeries message is an array of bytes.

In order to retain the flexibility that results from this, while making it fairly easy to parse FIX messages in and out of the Node, a simple language-neutral representation is used by the adapter. If you are programming entirely in Java there are helper methods in the [MQHelper](#) class which convert to and from the `IFIXMessage` message object used by Node and the "raw" MQSeries message.

The format of the MQSeries character-based message is a simple XML-style tag/value format as follows:

```
<fixMessage>
  <field>
    <key>[key-data]</key>
    <value>[value-data]</value>
  </field>
</fixMessage>
```

It is not necessary to populate the header and trailer FIX fields. The Node will populate these fields in the session layer before the message is sent. You need to populate the fields required by the message type you are sending - for example order quantity, symbol, side for a FIX NewOrderSingle. The one exception is that you will need to populate the TargetCompID header fields so the Node can route outbound messages to the matching FIX session.

There are three other types of message that may be placed on the queue by the IBM MQ Adapter:

Document Root Tag	Keys	Value
connectionStatusMessage	status	[true false]
exceptionMessage	exception	Exception details
resetMessage	N/A	N/A

## 7.6.4. Configuration

### Configuration Elements

There are two Catalys Node IBM MQ message processors:

- **IBMMQMessageListener** - listens for inbound FIX messages received on a session and writes to an MQSeries queue or topic.

For a complete list of attributes, see the [IBMMQMessageListener Configuration Reference](#).

- **IBMMQMessageSource** - reads messages from an MQSeries queue and sends them out a session.

For a complete list of attributes, see the [IBMMQMessageSource Configuration Reference](#).

### Single Session Configuration Examples

The following example shows an MQ Adapter configured on one session:

```
<Sessions>
  <Session counterpartycompid="EXCHANGE" compid="BROKER"
    fixversion="4.4" heartbeat="60">
    <Connections>
      <SocketConnection id="scl" hostname="localhost" port="2000"/>
    </Connections>
    <SessionManager>
      <SourceMessageProcessors>
        <IBMMQMessageSource queue.manager.name="QM_local"
          queue.name="EXCH_OUTBOUND"
          id="mqsource1"/>
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <IBMMQMessageListener queue.manager.name="QM_local"
          queue.name="EXCH_INBOUND"
          id="mqlisten1"/>
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
</Sessions>
```

### Multi-Session Configuration Examples

The MQ Adapter can be configured such that one queue is used to send messages out multiple FIX sessions. When this configuration is used, the FIX payload in the MQ message must contain valid TargetCompID field(s) that match one of the configured FIX sessions. The following example illustrates multiple sessions sharing the same queue:

```
<Sessions>
```

## Technology Adapters

```
<Session counterpartycompid="EXCHANGE_A" compid="BROKER" subid="Trader1"
  fixversion="4.2" heartbeat="60">
  <Connections>
    <SocketConnection id="idN65553" hostname="localhost" port="2000"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors>
      <IBMMQMessageSource queue.manager.name="QM_local"
        queue.name="EXCH_OUTBOUND"
        id="mq_source1"/>
    </SourceMessageProcessors>
    <ListenerMessageProcessors>
      <IBMMQMessageListener queue.manager.name="QM_local"
        queue.name="EXCH_INBOUND"
        id="mq_listen1"/>
    </ListenerMessageProcessors>
  </SessionManager>
</Session>
<Session counterpartycompid="EXCHANGE_B" compid="BROKER" subid="Trader2"
  fixversion="4.2" heartbeat="60">
  <Connections>
    <SocketConnection id="idN65587" hostname="localhost" port="2001"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors>
      <IBMMQMessageSource queue.manager.name="QM_local"
        queue.name="EXCH_OUTBOUND"
        id="mq_source2"/>
    </SourceMessageProcessors>
    <ListenerMessageProcessors>
      <IBMMQMessageListener queue.manager.name="QM_local"
        queue.name="EXCH_INBOUND"
        id="mq_listen2"/>
    </ListenerMessageProcessors>
  </SessionManager>
</Session>
</Sessions>
```

In this particular configuration, outbound messages put onto the "BUY\_OUTBOUND" queue must contain valid TargetCompID and TargetSubID fields. The Node will then use these two fields to decide which session to send the message out.

### 7.6.4.1. Templates

Fully working templates of a Node configured with the IBM MQ Adapter can be found in the *templates/MQAdapter* directory. The ReadMe.txt file located in each directory gives a full description on how to run these templates.

### Test Utility Classes

There are test utility classes available in the Node to test IBM MQ Series configurations. These are in the *com.camerontec.catalys.server.adapter.mq.test* package.



- **ReadandParseInboundMessageFromMQ** - this utility waits for messages on a MQ queue subject, when one is received, it transforms it into an `IFIXMessage` using the `MQHelper` provided, and then prints the message contents to `System.out`.

This can be invoked as follows:

```
java -classpath <MQ_JARS_AND_CATALYS_NODE_JAR%> \
    com.camerontec.catalys.server.adapter.mq.test.ReadandParseInboundMessageFromMQ \
    -q.manager <%MQ_QUEUE_MANAGER_NAME%> -q <MQ_QUEUE_NAME%>
```

- **SendOutboundFIXMMessagetoMQ** - this utility publishes a FIX message to a MQ queue every five seconds. It uses the `MQHelper` to transform the message from an `IFIXMessage` object to a raw character-based message and prints the message contents to `System.out`.

This can be invoked as follows:

```
java -classpath <MQ_JARS_AND_CATALYS_NODE_JAR%> \
    com.camerontec.catalys.server.adapter.mq.test.SendOutboundFIXMMessagetoMQ \
    -q.manager <%MQ_QUEUE_MANAGER_NAME%> -q <MQ_QUEUE_NAME%>
```

## 7.7. TIBCO Rendezvous Adapter

### 7.7.1. Introduction

The TIBCO Rendezvous adapter is designed to work with the Node and the TIBCO Rendezvous messaging product. Together with the Node, FIX messages can be sent to and received from any application connected to TIBCO Rendezvous.

### 7.7.2. Adapter Installation

TIBCO Rendezvous should be installed. See the TIBCO documentation. The TIBCO Rendezvous Adapter is a built-in message processor. It also requires the standard TIBCO Rendezvous .jar files containing the `com.tibco.tibrv.*` classes be added to the classpath.

The adapter is compiled against TIBCO Rendezvous 6.9, and is compatible with 7.2. Other Rendezvous versions may work with the adapter, but are not officially supported. Note that TIBCO always provides a JMS (Java Messaging Service) bridge for Rendezvous, so our [JMS adapter](#) will also work with all TIBCO versions.

### 7.7.3. Typical Configuration and Message Conversion

A typical scenario for message interaction between the Node and a TIBCO application might be:

1. A FIX counterparty establishes a socket connection with the Node

2. The Node writes inbound FIX messages to a queue or topic
3. The customer application reads inbound messages from the queue or topic
4. The customer application writes outbound messages to another queue or topic
5. The Node consumes messages from the queue or topic
6. The Node sends these messages out a FIX session to a counterparty

The TIBCO adapter does not assume that the application is written in Java. It can be written in any language that can communicate with the TIBCO Rendezvous technology.

Three standard FIX message formats are supported with the TIBCO adapter:

1. **TIBRVTransformer** - each FIX field is placed in a separate **TibrvField**. Each **Tibrv** field has a name that is the FIX field number it represents. The value of each field is the type and value of the FIX field.
2. **TIBRVNameTransformer** - is the same as the **TIBRVTransformer** except that **Tibrv** field names are FIX field names.
3. **TIBRVRawTransformer** - contains one field with the name "message", and the value is the entire FIX message.

Besides the FIX message type, there are three other types of messages that may be passed over the TIBCO adapter:

Message Type	Field Name	Value
Connection Status	Connection.status	[true false]
Reset	reset	reset
Exception	exception	Exception details

## 7.7.4. Configuration

### Configuration Elements

There are two Node TIBCO Rendezvous message processors:

- **TIBRVMessageListener** - listens for inbound FIX messages received on a session and writes to a TIBCO subject. For a complete list of attributes, see the [TIBRVMessageListener Configuration Reference](#).
- **TIBRVMessageSource** - reads messages from a TIBRV subject and sends them out a session. For a complete list of attributes, see the [TIBRVMessageSource Configuration Reference](#).

### Single Session Configuration

The following examples shows a TIBCO adapter configured on one session:

```

<Sessions>
  <Session counterpartycompid="EXCHANGE" compid="BROKER"
    fixversion="4.4" heartbeat="60">
    <Connections>
      <SocketConnection id="scl" hostname="localhost" port="2000"/>
    </Connections>
    <SessionManager>
      <SourceMessageProcessors>
        <TIBRVMessageSource subject="EXCH_OUTBOUND" id="tms"/>
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <TIBRVMessageListener subject="EXCH_INBOUND" id="tml"/>
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
</Sessions>

```

## Multi-Session Configuration

The TIBCO Adapter can be configured where one TIBRV subject can be used to send message out multiple FIX sessions. When this configuration is used, the TIBRV message is required to set the value of the targetCompID field(s) to the same values configured on one of the sessions.

```

<Sessions>
  <Session counterpartycompid="EXCHANGE_A" compid="BROKER" subid="Trader1"
    fixversion="4.2" heartbeat="60">
    <Connections>
      <SocketConnection id="idN65553" hostname="localhost" port="2000"/>
    </Connections>
    <SessionManager>
      <SourceMessageProcessors>
        <TIBRVMessageSource subject="EXCH_OUTBOUND" id="tms1"/>
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <TIBRVMessageListener subject="EXCH_INBOUND" id="tml1"/>
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
  <Session counterpartycompid="EXCHANGE_B" compid="BROKER" subid="Trader2"
    fixversion="4.2" heartbeat="60">
    <Connections>
      <SocketConnection id="idN65587" hostname="localhost" port="2001"/>
    </Connections>
    <SessionManager>
      <SourceMessageProcessors>
        <TIBRVMessageSource subject="EXCH_OUTBOUND" id="tms2"/>
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <TIBRVMessageListener subject="EXCH_INBOUND" id="tml2"/>
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
</Sessions>

```

```
</Sessions>
```

In this particular configuration, outbound messages put on "EXCH\_OUTBOUND" subject must contain TargetCompID and TargetSubID fields. The Node will then use these two fields to decide which session to send the message out on.

## 7.7.5. Templates

Working demonstrations of the TIBCO adapter can be found in *templates/TIBRVAdapter*.

### Test Utility Classes

There are test utility classes available in the Node to test TIBCO configurations. These are in the [com.camerontec.catalys.server.adapter.tibrv.test](#) package.

- **ReadandParseInboundMessageFromTIBRV** - this utility waits for messages on a TIBRV subject, when one is received, it transforms it into an `IFIXMessage` using the `TIBRVTransformer` provided, and then prints the message contents to `System.out`.

This can be invoked as follows:

```
java -Djava.library.path=<%PATH_TO_TIBRV_LIBS%> \
  com.camerontec.catalys.server.adapter.tibrv.test.ReadandParseInboundMessageFromTIBRV \
  <%TIBRV_SUBJECT_NAME%> <%TIBRV_SERVICE_NAME%> <%TIBRV_NETWORK_NAME%> \
  <%TIBRV_DAEMON_NAME%>
```

- **SendOutboundFIXMMessagetoTIBRV** - this utility publishes a FIX message to a TIBRV subject every five seconds. It uses the `TIBRVTransformer` to transform the message from an `IFIXMessage` object to the format described above and prints the message contents to `System.out`.

This can be invoked as follows:

```
java -Djava.library.path=<%PATH_TO_TIBRV_LIBS%> \
  com.camerontec.catalys.server.adapter.tibrv.test.SendOutboundFIXMMessagetoTIBRV \
  <%TIBRV_SUBJECT_NAME%> <%TIBRV_SERVICE_NAME%> <%TIBRV_NETWORK_NAME%> \
  <%TIBRV_DAEMON_NAME%>
```

Two more sets of send and receive utility classes that demonstrate the `TIBRVRawTransformer` and `TIBRVNameTransformer` are also available.

## 7.8. Microsoft MQ Adapter

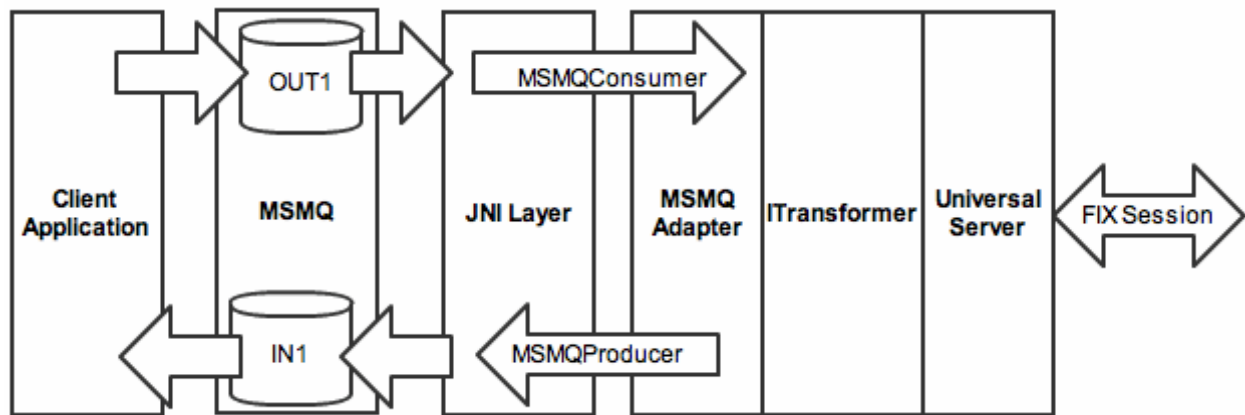
### 7.8.1. Introduction

The MSMQ Adapter is designed to work with the Node and Microsoft's Message Queuing platform (MSMQ). MSMQ is a messaging infrastructure and environment for running distributed, loosely coupled messaging applications in a Windows environment. Source MSMQ applications place messages on a queue, where they are temporarily stored, then a target MSMQ application consumes the messages from the queue for processing. Such applications can communicate across heterogeneous networks and with computers that may be offline temporarily.

Normally there are two dedicated queues used to distribute information to and from the Node. An "In" queue is where FIX messages sent to the Node are placed and an "Out" queue is where messages that are to be sent by the Node are placed.

### 7.8.2. Overview

The following diagram shows how messages are sent between a MSMQ client application and the MSMQ Adapter running on the Node.



Because there is no direct access to the MSMQ from Java, we use a JNI layer to create a bridge to the native Windows API.

- **Client Application** - application which will use the MSMQ to send and receive messages to/from the Node.
- **MSMQ** - This is the Windows messaging framework which manages the queues.
- **JNI Layer** - An interface to the MSMQ API contained in one of MSMQWrapper DLLs delivered with the Node.
- **MSMQ Adapter** - Technology adapter configured on the Node which sends/receives messages to the MSMQ.

- **ITransformer** - This [IMessageTransformer](#) transforms message between MSMQ message objects and Node `IFIXMessage` objects.

### 7.8.3. MSMQ Installation

Windows servers usually come with MSMQ pre-installed. If MSMQ is not installed, please review the Administrator Guide for the version of Microsoft Windows Server you are running.

#### MSMQ Queues

Queues to handle inbound and outbound messages from the Node will need to be created on the MSMQ server. These queues should give full access to the user which the Node is running under.

To create a private MSMQ queue for the example, you need to open **Computer Management**, then select **Services and Applications**, click **Message Queuing**, right-click on **Private Queues** and select **New**.

#### MSMQ Wrapper DLL

The requisite 32-bit and 64-bit MSMQ wrapper DLLs (*MSMQWrapper32.dll* and *MSMQWrapper64.dll*) are bundled with the Node in the *lib* directory of your installation. If they are not present, then re-run the Node installer and select the MSMQ option.



#### Important

Choose the appropriate DLL for your Windows server and rename it to *MSMQWrapper.dll*. The Node will only look for a DLL with that name.

In order for the Node to reference the *MSMQWrapper.dll* you will need to put the DLL in your system path (e.g. *C:\Windows*) or you will need to set the `-Djava.library.path` Java system property in the startup command to the directory where the DLL is located. For example:

```
java -Djava.library.path=D:\FIX\catalys-node\lib\ ...
```

### 7.8.4. Configuration

The MSMQ Adapter does not assume that the client application is written in Java — it can be written in any programming language that can communicate with MSMQ. By default the body of the MQ messages will contain a raw FIX message for both sending and receiving.

## Configuration Elements

There are two Node MSMQ message processors:

- **MSMQMessageListener** - listens for inbound FIX messages received on a session and writes to an MSMQ queue. For a complete list of attributes, see the [MSMQMessageListener Configuration Reference](#).
- **MSMQMessageSource** - reads messages from an MSMQ queue and sends them out a session. For a complete list of attributes, see the [MSMQMessageSource Configuration Reference](#).

## Configuration Example

The following example shows an MSMQ Adapter configured on one session:

```
<Sessions>
  <Session counterpartycompid="EXCHANGE" compid="BROKER"
    fixversion="4.4" heartbeat="60">
    <Connections>
      <SocketConnection id="scl" hostname="localhost" port="2000"/>
    </Connections>
    <SessionManager>
      <SourceMessageProcessors>
        <MSMQMessageSource queue.formatName="DIRECT=OS:.\private$\out1"
          queue.isTransactional="false"
          id="msmq_s"/>
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <MSMQMessageListener queue.formatName="DIRECT=OS:.\private$\in1"
          queue.isTransactional="false"
          id="imsmq_l"/>
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
</Sessions>
```

## Template

There is a working template in the *templates/MSMQAdapter* directory which demonstrates two Node instances exchanging messages via MSMQ. Refer to this template's documentation for full details on how to run it.

### 7.8.5. Troubleshooting

- Ensure that you renamed either the *MSMQWrapper32.dll* or *MSMQWrapper64.dll* to *MSMQWrapper.dll*. The Node will only look for a DLL with that name.
- Ensure that the *MSMQWrapper.dll* is either in your system path (e.g. in *C:\Windows*) or its directory is referenced in the startup command using the `-Djava.library.path` Java property.

## 7.9. JDBC Adapter

### 7.9.1. Overview

The Node JDBC Adapter is designed to work with the Node and any JDBC compliant database product.

The JDBC Adapters provides communication and exchange of messages between a database and the Node. When placed in the listener (inbound) side of a session, it extracts a selection of fields and inserts them into a database using a SQL statement or stored procedure. Values may be converted to specific data types or will default to `varchar`s.

On the source (outbound) side of a session, the adapter periodically polls a database using the a SQL statement or stored procedure. The returned row(s) are used to populate fields of a FIX message object. The message is sent out on a session. A `status` column is updated depending on if the message was successfully sent or not.

The same instance of the JDBC adapter can act both as a source and a listener when defined in both sections, with all the attributes defined in the first section and just the same id in the second.

### 7.9.2. JDBC Adapter Installation

You must have your JDBC provider's software according to their installation instructions. Typically, you will be connecting to a JDBC compliant database. You will need to locate the `.jar` file(s) for the JDBC class library of the database and place them in the `/ext` directory of your install.

You will need to locate your JDBC driver and configure the `driver` attribute on the JDBC adapter to the driver's Java class name. For example: `driver="sun.jdbc.odbc.JdbcOdbcDriver"`.

### 7.9.3. Configuration

#### JDBCAdapter element

The `JDBCAdapter` element is a message processor which is configured into the source and listener message processor chains of a session on the Node.

For a complete list of attributes, see the [JDBCAdapter Configuration Reference](#).

#### Configuration Example

The following examples shows a JDBC Adapter configured on one session:

```
<Sessions>
  <Session counterpartycompid="EXCHANGE" compid="BROKER"
    fixversion="4.4" heartbeat="60">
```



```

<Connections>
  <SocketConnection id="sc1" hostname="localhost" port="2000"/>
</Connections>
<SessionManager>
  <SourceMessageProcessors>
    <JdbcAdapter
      id="jdbc1"
      driver="com.mysql.jdbc.Driver"
      url="jdbc:mysql://localhost/jdbcadapter"
      user="user"
      password="password"
      fixToDbFile="../../tuples4.txt"
      fixToDbFileFormat="desc,tag"
      fixToDbInsertStatement="INSERT INTO from_fix VALUES"
      peerRecoveryInterval="10000"
      peerPollingInterval="1000"
      dbToFixTags="ignore,ignore,ignore,55,35"
      dbToFixPollStatement=
        "SELECT * FROM to_fix WHERE to_fix.status < '2' ORDER BY to_fix.msgId ASC"
      dbToFixUpdateStatement=
        "UPDATE to_fix SET to_fix.status = ? WHERE to_fix.msgId = ?"
      dbToFixUseStrictSequencing="true"
      dbToFixInitStatement=
        "SELECT MAX(msgId) FROM to_fix WHERE to_fix.status > '1'"
    >
  </JdbcAdapter>
</SourceMessageProcessors>
<ListenerMessageProcessors>
  <MessageProcessorReference id="listen_jdbc1" refid="jdbc1"/>
</ListenerMessageProcessors>
</SessionManager>
</Session>
</Sessions>

```

## 7.9.4. Template

There is a working template for the JDBC Adapter in *templates/JdbcAdapter* directory. This is a simple template with a single FIX session where one side of a FIX session connects and sends orders to the other side side. The receiving side inserts these messages into a database table by assigning a relationship from a FIX fields to database column. At the same time, this side is selected data from a database, constructing FIX messages and sending them outbound on the session.

There are two variants of the sell side. One uses prepared SQL statements, the other uses stored procedures. The `config_sell` instance uses prepared SQL statements, while the `config_sellProc` instance uses stored procedures to interact with the database.

The JDBC Adapter template requires that you create tables and optionally stored procedures in your database. There are MySQL formatted DDL scripts for both the SQL statement and stored procedure templates (*createMySql.ddl* and *createMySqlProc.ddl*). These DDL scripts correspond to the `config_sell` and `config_sellProc` templates respectively.

See the *ReadMe.txt* file for the instance you are running for instructions on how to run the template.

## 7.9.5. Repeating Groups Considerations

If your FIX messages contain repeating groups you must ensure that your database schema has enough columns to accommodate all possible repeating group entries in your messages, otherwise the adapter will throw a SQL Exception.

You must also ensure that the list of tags specified via the attribute *fixToDbTags* and the tuples listed in the file specified via attribute *fixToDbFile* contain enough placeholders to accommodate as many repeating group entries as can possibly occur in your FIX messages. If a message contains more repeating groups entries than the available placeholders the adapter will throw a SQLException. If the message contains less entries the corresponding DB columns will simply be set to NULL.

## 7.10. Flat File Adapter

### 7.10.1. Introduction

The Flat File Adapter is designed to pass messages between the Node and a set of files. Messages received from FIX sessions are written to a file-based FIFO queue for reading by another system or application. In the other direction, the Flat File Adapter reads messages from a separate file-based FIFO queue and sends them out a FIX session.

### 7.10.2. Operation

The Flat File Adapter uses three FIFO file persistent queues; one for messages received from FIX, one for messages to be sent to FIX, and one (optionally) for messages which cannot be sent to FIX. The adapter is thus both a queue writer (messages from FIX are written to a `fromFixQueue`, messages which cannot be sent to FIX are optionally written to a `toFixErrorQueue`) and a queue reader (messages in the `toFixQueue` are read and sent to FIX).

It is the responsibility of the other party to:

1. Act as a queue reader of the `fromFixQueue`, adhering to the rules acting as a FIFO file queue reader.
2. Act as a queue writer to the `toFixQueue`, adhering to the rules for acting as a FIFO file queue writer.

Before sending a message out a session, the adapter marks the file as read in the `toFixQueue`. If the flat file adapter suffers an outage while attempting to send the message to FIX and before the message is removed from the queue, the message state is retained in the queue. On recovery, the flat file adapter determines that the last message has already been read and explicitly sets the `PossResend` flag to indicate a possible resend to the other party.

If the `toFixErrorQueue` is enabled, messages which cannot be sent to a session are written to it. In this case it is the responsibility of the other party to read the `toFixErrorQueue` to detect such errors.

## 7.10.3. Configuration

The adapter is configured by adding a `FlatFileAdapter` element inside both the `SourceMessageProcessors` and `ListenerMessageProcessors` elements of a session. The `FlatFileAdapter` element must be present in both the source and message listener chain, and both configuration instances point to the same runtime element. This is configured by having a `MessageProcessorReference` in one of the message processing chains referring to the other.

For a complete list of attributes, see the [FlatFileAdapter Configuration Reference](#).

The following example shows a `FlatFileAdapter` element configured on one session with the minimum attributes configured:

```
<Sessions>
  <Session counterpartycompid="EXCHANGEA" compid="BROKER"
    fixversion="4.2" heartbeat="60">
    <Connections>
      <SocketConnection id="scl" listenport="2000"/>
    </Connections>
    <SessionManager>
      <SourceMessageProcessors>
        <FlatFileAdapter id="ffa"
          fromFixQueue="./fromFix"
          toFixQueue="./toFix"
          toFixErrorQueue="./toFixError"
          peerPollingInterval="10"/>
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <MessageProcessorReference id="listen_ffa" refid="ffa"/>
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
</Sessions>
```

## 7.10.4. File System Persistent Message Queue

### Specification

The file system persistent message queue supports FIFO semantics via the naming of message files. That is, each new message is placed in a separate file and given a name which represents its relative position in the queue. Thus the name of each message file contains a queue sequence number. Note that the queue sequence number may not always correspond with the FIX sequence number of the contained message.

## Rules For Acting As a FIFO File Queue Writer

Queue writers must guarantee that the next message written has a queue sequence number greater than any existing message files. Queue readers must guarantee to read the message file with the lowest queue sequence number.

In order to safely survive process and/or host outages queue writers must also guarantee that all messages on the queue are not corrupted during the write phase. A two-phase write is used to achieve this:

1. **Write Phase** - when a new message is to be written, the queue writer chooses a message file name according to the above rule and writes the message to a temporary file.
2. **Commit Phase** - when the entire message is written to the temporary file, the writer closes the temporary file and renames it indicating that it is now on the queue.

If there is an outage during the write phase, then the message is never placed on the queue. The rename performed in the commit phase is atomic with respect to the file system and therefore is a message file makes it to the message queue it is guaranteed not to be corrupt.

Note that on some platforms, a queue reader may be able to see that a message file is on the queue but not (yet) be able to read it. This can occur if the readers attempts to open the file during the commit phase. In this case the reader should try again after a suitable delay.

## Rules For Acting As a FIFO File Queue Reader

Queue readers must guarantee to:

1. Ignore temporary message files (since they are not yet on the queue)
2. Read the message file with the lowest queue sequence number

Note that step 2 above implies that the reader **must not** maintain any state with respect to the state of the queue since:

- Queue writers are responsible for the naming of queue files and therefore the allocation of queue file sequence numbers.
- A queue write may suffer an outage and subsequently recover while the queue is empty and the reader is waiting for the next message. In this case the queue writer loses all information about the state of the queue and on the next write will choose the lowest available sequence number. That is, the queue sequence number is effectively reset.

Queue readers must handle this sequence of events by always choosing the next message with the lowest available sequence number regardless of the sequence number of the most recently read message.

When a queue reader has processed a message file it must remove the message file from the queue. This is achieved by renaming the file to indicate that the file is no longer on the queue. Note that message files **must not** be removed from the queue until processing is complete.

Queue readers may optionally mark message files as read after reading, but before removal. This can aid a reader's recovery processing after an outage in case there is a restriction on duplicate message processing. The flat file adapter uses this mechanism to set the `PossResend` flag if it detects the possibility of sending a message more than once.

## 7.10.5. File Naming Conventions

Each individual FIFO file queue exists in its own dedicated directory. Communication between parties via a single queue may occur in one direction only. That is, no party should be both a reader and a writer of the same queue.

Each message in the queue is contained in a separate file in the queue directory and is distinguished by its name. The name of each contains two pieces of information about the file:

- Its state
- Its relative position in the queue.

The state of the message is encoded via a single character prefix according to the following table:

File Name Prefix	File State
T	Temporary file in the process of being written and is not yet in the queue. The queue writer creates and writes to this file.
Q	The file is in the queue and is available for reading. The queue writer creates this file by renaming a temporary file.
R	The file is in the queue but has been read by the queue reader. This is an optional state and is controlled by the queue reader, however, the queue writer must regard this as valid queue file. There should be at most one file with this prefix in the queue at any time.

A file's relative position in the queue is specified by its sequence number suffix. The lower the sequence number, the closer to the head of the queue. The higher the sequence number, the closer to the tail of the queue. The queue writer determines the sequence number of the next file it writes.

## 7.10.6. Queue Snapshots

The following tables show some typical queue snapshots at various stages of processing:

Queue contents	Explanation
T1	The queue writer is in the process of writing the first message file to the queue.

Queue contents	Explanation
Q1	The queue writer has written the first message file and placed it on the queue. It is now available for reading by the queue reader.
T2	The queue writer is in the process of writing the second message file to the queue.

Queue contents	Explanation
R1	The queue reader has read the first message file and is processing it. The message is still on the queue.
Q2	The queue writer has written the second message file and placed it on the queue.
Q3	The queue writer has written the third message file and placed it on the queue.
T4	The queue writer is in the process of writing the fourth message file to the queue.

Queue contents	Explanation
R4	The queue reader has read, processed and deleted the first three entries from the queue and has read and is processing the fourth entry.

Queue contents	Explanation
Q1	The queue writer has recovered from an outage while the queue happens to be empty and has

Queue contents	Explanation
	written another message to the queue. Note that the sequence number has been reset. The queue reader must cope with this situation by always reading the message file with the lowest sequence number, regardless of the sequence number of the previously read file.

## 7.10.7. Templates

See *templates/FlatFileAdapter* for a working example of this adapter.

## 7.11. RMI Adapter

### 7.11.1. Introduction

The RMI Adapter is designed to work with the Node and any RMI enabled application. The adapter is designed to work in a client/server environment. The server is the Node and the clients are the RMI enabled applications. Many RMI enabled applications can send messages through a single Node.

### 7.11.2. Configuration

The `RmiAdapter` element is configured as either a source or listener message processor, and then referenced by the other message processing chain using a `MessageProcessorReference` element. The RMI Adapter can also be used across multiple sessions by providing a `MessageProcessorReference` to the original RMI Adapter instance in both the source and listener message processing chains.

For a complete list of attributes, see the [RmiAdapter Configuration Reference](#).

## Configuration Examples

This example shows the RMI Adapter configured on one session:

```
<Sessions>
  <Session counterpartycompid="EXCHANGE1" compid="BROKER" fixversion="4.2" heartbeat="60">
    <Connections>
      <SocketConnection listenport="2000"/>
    </Connections>
    <SessionManager>
      <SourceMessageProcessors>
        <RmiAdapter id="rmi" name="session1" port="1099"/>
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <MessageProcessorReference id="rmi_listen" refid="rmi"/>
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
</Sessions>
```

```

    </ListenerMessageProcessors>
  </SessionManager>
</Session>
</Sessions>

```

The following XML configuration shows an example of the multi-session case.

```

<Sessions>
  <Session counterpartycompid="EXCHANGE1" compid="BROKER" fixversion="4.2" heartbeat="60">
    <Connections>
      <SocketConnection id="sc1" hostname="localhost" port="2000"/>
    </Connections>
    <SessionManager>
      <SourceMessageProcessors>
        <RmiAdapter id="rmi" name="session1" port="1099"/>
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <MessageProcessorReference id="rmi_listen" refid="rmi"/>
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
  <Session counterpartycompid="EXCHANGE2" compid="BROKER" fixversion="4.2" heartbeat="60">
    <Connections>
      <SocketConnection id="sc2" hostname="localhost" port="2001"/>
    </Connections>
    <SessionManager>
      <SourceMessageProcessors>
        <MessageProcessorReference id="rmi2" refid="rmi"/>
      </SourceMessageProcessors>
      <ListenerMessageProcessors>
        <MessageProcessorReference id="rmi2_listen" refid="rmi"/>
      </ListenerMessageProcessors>
    </SessionManager>
  </Session>
</Sessions>

```

## RMI Adapter Persistent Collection Properties

When the `directoryName` attribute is set, messages received from a session are persisted so they can be retrieved by an RMI client. Internally the RMI adapter uses the Node persistent collections framework which allows for an extra level of configuration regarding the underlying persistence mechanism. See the [Persistent Collection Properties](#) for more details of the available attributes. Specifically, the [Map](#)-related attributes are relevant.

These attributes are configured with a nested `Properties` elements as shown:

```

<RmiAdapter id="rmi" name="session1" port="1099">
  <Properties id="RmiAdapterPersistenceProperties">
    <Property name="isCachingEnabled" value="false"/>
  </Properties>
</RmiAdapter>

```



If this attribute is specified then the persistence files must be compacted periodically. This is done as part of the component's end of day procedure, which is invoked by `EndOfDay` task in the Scheduler or by issuing the [s\\_end\\_of\\_day](#) CLI command.

### 7.11.3. Catalys Node Object Model

The object model is simple but powerful. The essential features of the model are:

- A `Session` object is used to send and receive `Message` objects.
- `Session` objects can be listened to by `FixListeners`.
- `FixListeners` are notified when a `Session` receives a `Message`, raises an exception, connects or disconnects, or has its sequence numbers reset.
- Multiple `Message` objects can be represented by a `MessageList`.
- Each `Message` object has an ordered collection of `Field` objects.
- Each `Field` consists of a tag (the FIX tag for the field) and a value.
- A tag/value pair can be held in a `NameValue` object.

Those are the common objects you will use. There is also:

- The `Application` object represents the Node itself.
- The `Application` object provides a method for connecting to known `Sessions`.
- The `Application` also provides a `MessageFactory` object which is used to create new `Message`, `MessageList` and `Field` objects.

Here are the steps your RMI application may go through to connect to a Node and send and receive messages:

1. Connect to a running Node by creating an `Application` object.
2. Get the `MessageFactory` from the `Application` object.
3. Get the `Session` you want to communicate with from the `Application`.
4. Create a `FixListener` object which contains your code for processing events (messages, resets, exceptions etc.) from this `Session` object.
5. Use the `MessageFactory` to create a `Message` object.
6. Use the `Message` methods to populate its fields.
7. Send the `Message` object using the `send` method.

### 7.11.4. Class Summary

For a complete list of RMI Adapter classes, interfaces and their associated methods please review the [RMI Package Summary](#).



## Note

Some objects have multiple methods which provide access to essentially the same data. The difference is where the data itself resides: locally or on the Node. Method calls on local objects are much faster than calls on remote objects which, necessarily, go across RMI. The Node provides you with both options. An example is the `getFieldObjects` and `getTagValues` methods on message objects. Both methods give you access to the message's field data but the former gives you access through remote `IField` objects while the latter gives you access through local `NameValue` objects.

## 7.11.5. Templates

Start with the supplied templates and sample configuration files in *templates/RMIAdapter*.

There are three templates in this directory:

- **Basic** - the [TestRmi](#) client sends messages to one session, whose parties can be specified on the command line.
- **GetMessage** - the [TestGetMessage](#) client receives messages from one session, whose parties can be specified on the command line.
- **MultiSession** - the [TestMultiSessionRmi](#) client sends messages to multiple sessions, whose parties are detected automatically.

## Running the RMI Client Applications on a Separate Machine

The templates assume that the RMI client application is running on the same machine as the Node.

They can also run on different machines on the same network. The sample RMI test programs require the full RMI name of the server, including the machine name and RMI registry port (if it is different from the default). A full RMI name is in the form:

```
//[<host>][:<port>]/<instance_name>
```

The sample RMI programs take an RMI name as a parameter. For example:

```
java com.camerontec.catalys.server.adapter.rmi.test.TestRmi //server1:1099/CatalysRouter
```

## 7.11.6. Troubleshooting

Before looking at the problem in detail, check that:

- The Node is running
- The RMI client application is running
- The required jar files are present on the Java classpath
- The Node and RMI client application are able to access each other over the network

If all of the above conditions are satisfied, then use the following table to diagnose the problem:

Behaviour	Diagnosis
<code>java.rmi.ConnectException: Connection refused to host</code>	<p>Possible causes:</p> <ul style="list-style-type: none"> <li>• The Node is not running or had errors during startup</li> <li>• No <code>RmiAdapter</code> element is configured on the Node</li> <li>• The Node machine is not visible to the RMI client application</li> </ul>
<code>java.rmi.ServerException: RemoteException</code>	<p>The Node threw an exception, possible causes:</p> <ul style="list-style-type: none"> <li>• Session is not logged on or connected</li> <li>• Session encountered an <code>IOException</code> while trying to send a message</li> <li>• Some other problem performing a remotely called method</li> </ul> <p>The actual <code>RemoteException</code> object contains details of the nature of the problem. The real cause of the problem will often be revealed by one of the "nested exceptions". These nested exceptions are available programmatically by examining the <code>RemoteException</code> object.</p> <p>These exceptions may also go back to your program for processing through the <code>onException</code> call back method on any listener(s).</p>

Behaviour	Diagnosis
[any kind of exception] ...at sun.rmi.transport.StreamRemote- Call.exceptionReceivedFromServer ...	<p>This indicates that your application called a method on one of the remote Node objects. The call reached the Node object but the object encountered an exception while trying to execute the method.</p> <p>These exceptions occur at a low level and therefore may not be sent back to your listener's <code>onException</code> methods.</p> <p>Possible causes:</p> <ul style="list-style-type: none"> <li>• Bad configuration - missing jar files on classpath</li> <li>• The Node's JVM has run out of some resource, for example memory.</li> </ul>
The RMI Application pauses periodically, impacting performance	<p>By default, RMI does a full garbage collection every hour. To switch this off (and avoid hourly application pauses), set the <code>sun.rmi.dgc.server.gcInterval</code> Java system property to the maximum value.</p>

## 7.12. EMX Adapter

### 7.12.1. Introduction

The Catalys Node EMX Integrator is a complete package that will connect to the EMX mutual fund trading service. It is developed by ReadySolve (<http://www.readysolve.com/>) and distributed by CameronTec.

Most EMX Integrator clients use the Node to run their FIX engine. This permits a stand alone and independent server to make a connection to EMX and to accept and present the messages through a technology adapter. CameronTec has a selection of technology adapters available and new ones can easily be written to connect to any message source.

The EMX Integrator is also available as a set of Java class libraries that enable a Java application developer to connect to EMX without needing to attend to the details of the FIX protocol or, in particular, the particular FIX protocol modifications that the EMX service requires. In this form it is ideal for embedding into integrated systems.

Whether you are using the Catalys Node Server or writing a Java class that extends the EMX Integrator's base class, your first task is to install the EMX Integrator and get the demonstration programs running

as described below. You can then write whatever program code you need to link the EMX Integrator to your existing computer systems. This may possibly be none at all, if you use the Catalys Node Server and choose the [EMX Delimited File Adapter](#) as your technology adapter (which used the same format of tab-delimited files as the EMX web site's file upload/download facility). Finally, you need to test your programs against Euroclear's EMX trialling system and to go through their compliance procedures.

Most users of the EMX Integrator need only refer to this documentation and a few parts of the core Catalys Node documentation. Those using the Catalys Node Server will need to refer to its documentation as well.

For reference, the Catalys Node components that comprise the EMX Integrator are:

- The Catalys Node EMX Add-On, described here.
- The Catalys Node core implementation of the [FIX](#) protocol.
- The [Catalys Node SSL](#) Secure Socket Layer implementation.
- The [Catalys Node Session Persistence](#).
- The Catalys Node Server documentation (if being used).

Your EMX Integrator installation also includes:

- Sample working programs with full source code in the `templates` directory.

## 7.12.2. Introducing the EMX Integrator

The EMX Integrator provides the following:

- For those developers who wish to have an in-process FIX engine that will connect to EMX, we provide a base class, [EMXClient](#), for you to extend. This base class will connect to the EMX system, maintain a connection and manage all of the message traffic. Your extension of the base class will define the actions that should be taken when an EMX message is received and will generate outgoing EMX messages that will be passed to the base class for sending to the EMX Hub.
- For those who wish to use the Catalys Node Server to connect to EMX, we provide a ConnectionManager, [EMXConnectionManager](#), which plugs in to the Catalys Node Server in place of the standard ConnectionManager. It does everything that the Catalys Node Server's standard ConnectionManager does (in fact, it is an extension of it) but has additional behaviour to support EMX's protocol.
- Support for SSL with Client Authentication, which EMX requires.
- Signing and verification of all EMX messages, according to EMX's custom message signing protocol. This is built in to the [EMXClient](#). For users of the Catalys Node Server, we provide MessageProcessors that perform the signing and verification of EMX messages. Your Catalys Node Server configuration file just needs to indicate that these MessageProcessors should be used and the Catalys Node Server will use them automatically.

- Support for EMX's Mutual Fund FIX messages, implemented through an [EMXMessage](#) class that implements the Catalys Node [IFIXMessage](#) interface.

Two sample programs are provided, to illustrate the use of the EMX Integrator. These are:

- [com.camerontec.catalys.server.adapter.emx.test.DemoIFAClient](#)
- [com.camerontec.catalys.server.adapter.emx.test.DemoProviderClient](#)

There is also a configured Catalys Node Server demonstration application. These demonstration programs are [described below](#).

Users of the Catalys Node Server will not need to refer to the detailed documentation of the underlying Java classes, although some may want to. Users of the EMXClient toolkit approach will certainly need to, however. Full on-line documentation is provided for the complete set of Java classes that comprise the EMX Add-On. The most useful classes are:

- [com.camerontec.catalys.server.adapter.emx.manager.EMXClient](#).
- [com.camerontec.catalys.server.adapter.emx.message](#) package.
- [com.camerontec.catalys.server.adapter.emx.configuration.EMXConstants](#).

There are also some further classes, which deal with message signing and key management:

- [com.camerontec.catalys.server.adapter.emx.security](#) package.

## 7.12.3. Installation

You must have Java 8 or later installed.

The EMX Integrator classes are bundled with the Catalys Node package. This package needs to be installed before you can run the EMX Integrator.

## 7.12.4. Running the Demonstration Programs

### 7.12.4.1. First Steps

Demonstration programs are provided for you, to get you up and running quickly. You should ensure that these programs are running before attempting anything else.

There are four demonstration applications. All users should get the first three programs working first. The final demonstration is provided for Catalys Node Server users only and need not be used by those clients who are not using the Catalys Node Server.

1. A "Basic EMX Hub Emulator", that you can connect your programs to during your development and initial testing phases.
2. A Fund Provider demonstration, that responds to order and request messages.

3. An Intermediary demonstration, that generates order and request messages and displays the responses to them.
4. For clients that are using the Catalys Node Server as an Intermediary, there is a configuration file that will make a connection to the Basic EMX Hub Emulator and allow you to send and receive messages in EMX's standard upload/download format.

The demonstrations are ready to run. You will be starting three concurrent programs, which will connect with each other in this way:

**Table 7.1.**

DemoIFA- Client (CAFIX)	-->--> <--<--<	EMX Hub Emulator (EmuEMX)	-->--> <--<--<	DemoProvider- Client (CBFIX)
----------------------------	----------------	---------------------------------	----------------	---------------------------------

You will be using an emulator of the EMX Hub during your demonstration and testing phases, since Euroclear will not permit systems under development to connect to their trialling EMX Hub system.

Note: Euroclear discourages clients from using the EMX Client Test system until the client's system is ready for formal compliance testing. Except by prior arrangement, you will be allowed only to log on, log out and send FIX administration messages. If you try to send any EMX application messages they may be ignored. But proving that you can log on and maintain a FIX session with EMX is still a worthwhile first step to take. You will need to verify that your firewall settings have been correctly modified to enable you to connect to the EMX service, for example. When you have succeeded with this, and are using your own Participant IDs, you can ask Euroclear to allow you to send EMX application messages at some agreed time in the future.

#### 7.12.4.1.1. Locating the Demonstration Programs

Demonstration programs are provided with Catalys Node. You can and should test that you have installed Catalys Node correctly by running the EMX Integrator's demonstration programs before you commence any of your own development work.

The demonstration programs are found in the Catalys Node's `templates/EMX` subdirectory. As well as runnable demonstration programs, we include the source code for the demonstration programs themselves in `templates/src`. You are welcome to copy this source code to use as a starting point for developing your own system.

The `templates/EMX` directory contains the following subdirectories:

- `config`: the EMX adapter configuration
- `data`: the location of the incoming and outgoing message files used by the [EMX Delimited File Adapter](#)
- `log`: application log files
- `msg`: message log files

- `security`: 1024-bit and 2048-bit key and trust stores
- `seq`: sequence number log files
- `template`: location of file format templates used by the [EMX Delimited File Adapter](#)
- `unix`: scripts to run templates on unix
- `windows`: scripts to run template on windows

### 7.12.4.1.2. Running the Demonstration Programs

*Note: We use the Windows examples in this tutorial. The Unix examples can be run in a similar fashion.*

You should open three command windows, each with `templates/EMX/windows` as the current working directory.

In one window, type the command " `runEmuEMX` ". You should see the EMX Hub Emulator start up and shortly it will tell you "Listening on port 4433...". It is waiting for an EMX client program to connect to it on that port. You will also see a "Command:" prompt, which you should ignore for now. We will now connect two different clients to our EMX Hub Emulator.

In another window, type the command " `runCBFIX` ". This command starts up an instance of the DemoProviderClient program which has been configured with the EMX Participant ID of "CBFIX". The Participant ID "CBFIX" has been configured as an EMX Mutual Fund Provider service. You should see that this program connects to the EMX Hub Emulator and begins to exchange FIX administration messages. If this fails to occur then there is an error in your installation and you should contact the CameronTec support team.

In the third window, type the command " `runCAFIX` ". This command starts up an instance of the DemoIFAClient program which has been configured with the EMX Participant ID of "CAFIX". The Participant ID "CAFIX" has been configured as an EMX Mutual Fund Intermediary service. You should again see that this program connects to the EMX Hub Emulator and begins to exchange FIX administration messages. You are ready to exchange EMX application messages between the two client programs via the EMX Hub Emulator.

### 7.12.4.1.3. Exchanging Messages

Both of the client demonstration programs have a GUI window which shows all EMX application messages that are sent and received and allows you to control the client program. These windows do not show the FIX administration messages, only the EMX application messages, so at present they will both be empty.

Go to the CAFIX EMXMessageLister window and press the "Send Message" button. This button pops up a list of pre-prepared EMX application messages from which you can choose. Select the one labelled "TRFQ" and press the "Send" button. You will see the message be sent to the EMX Hub Emulator (EmuEMX) (look at the EmuEMX command window), and shortly afterwards you will get an acknowledgement `MAACK` from EmuEMX. At the same time, EmuEMX will forward the message to your



DemoProviderClient FIX engine (CBFIX) that you have previously connected. You will see the `TREQ` message in the CBFIX EMXMessageLister window and logged in the CBFIX command window.

The DemoProviderClient (CBFIX) is programmed to respond to the `TREQ` message by returning a `TRES` message, and by this time it will have done so. You will see its `TRES` response listed in the CBFIX EMXMessageLister window under the `TREQ` message. You will see the same in the CAFIX EMXMessageLister window, since the `TRES` message is forwarded by EmuEMX to the FIX engine that originated the request, CAFIX.

You will also see, in each case, a `MACK` message (an EMX Acknowledgement message). But these are not the same `MACK` message — they are two different ones. The EMX Hub Emulator, just as the real EMX Hub does, responds to each EMX application message that it receives with a `MACK`, if the message is valid and will be forwarded, or a `MERR`, if the message is in error and will not be forwarded.

If this `TREQ` demonstration sequence has completed correctly you may wish to try sending any of the other pre-prepared messages that are available from the list popped up by the "Send Message" button. The Intermediary client (CAFIX) can also send `OINP` and `VREQ` messages, while the Provider client (CBFIX) can initiate a `ECNP` message, as well as responding to any of the messages that the Intermediary client sends.

Playing around with sending a few messages will help you get a feel for how the EMX protocol works. So that you know what is going on, we would like to make you aware of these items:

- When you send any valid message, you will first receive an acknowledgement from EMX (the Emulator in this case) which simply tells you that EMX has received your message and will forward it to the target participant. The `MACK` acknowledgement does not promise that the target participant will be prepared to process the message.
- When a Fund Provider receives a message, its system must reply with a `MERR` message if it is unable to process the message or it rejects the message for any reason. If it accepts the message it will send a positive response (maybe after a short processing delay).
- Order messages (`OINP`) that are accepted by the Fund Provider will first receive a `TBKD` message (transaction booked), which is a promise that the Fund Provider will execute the order. But since an order is normally not executed in full until after the Fund Provider's next fund repricing cycle, the electronic contract note (`ECNP`), which contains full details of the processing of the order, will follow some time later. Our DemoProviderClient implementation emulates this by sending the `TBKD` response immediately while it batches up the sending of `ECNP` messages and sends any outstanding ones every two minutes.

### 7.12.4.2. A Catalys Node Server Demonstration Example

The final demonstration program, `runCAFIX_DFA`, is for use by Catalys Node Server clients in place of the DemoIFAClient demonstration program.

If you are planning to build your application in Java, using the Catalys Node's class libraries, then you do not need to run this demonstration program.

However, if (like the majority of our clients) you are planning to use the Catalys Node Server, you should run this demonstration program so that you can be sure that the Catalys Node Server is operating correctly and because you can, in due course, use the demonstration configuration file as a template for your own Catalys Node Server configuration file.

Once you have the above demonstration working, shut down the CAFIX client (select "Logout" from the menu, then "No" at the reconnect prompt), then run the example `runCAFIX_DFA` instead. This demonstration allows you to upload messages from data files, using the [EMXDelimitedFileAdapter](#). The demonstration is configured to upload any files that you put in the subdirectory `data/CAFIX/outgoing` and to write incoming messages into files that will be placed in the directory `data/CAFIX/incoming`. A source of data files has been provided for you in the directory `data/CAFIX/resources`. You can copy (drag and drop with the `Ctrl` key pressed if using Windows) files from `resources` to `outgoing` and the EMX Integrator will immediately begin sending the messages from the file.

If you are planning to use the EMX Delimited File Adapter as your interface to EMX then you have only to copy this working example and its configuration file to a sensible permanent location on your computer system and make small modifications to the template configuration file and command file. If you are planning to connect your system to the Catalys Node Server using any of the other Catalys Node technology adapter, you should also start by doing this. Then you will need to replace those parts of the template configuration file that refer to the `EMXDelimitedFileAdapter` with the proper configuration for your choice of technology adapter. You will need to refer to the documentation for your chosen adapter.

### 7.12.4.3. The Purpose of the Demonstration Programs

Running these demonstration programs will ensure that the Catalys Node has been installed correctly on your system. If, at any later stage, you become unsure that your installation is still working correctly, you can (and should) run the demonstration programs again as a check.

After you have the demonstration example programs working, you have a firm base from which to begin your own system development. If developing an Intermediary application, for example, you can leave the EMX Hub Emulator (EmuEMX) and the DemoProviderClient (CBFIX) in operation but replace the DemoIFAClient (CAFIX) with your own system under development.

### 7.12.4.4. The Limitations of the Demonstration Programs

The EMX Hub Emulator (EmuEMX) does very minimal validation. Just because EmuEMX accepts a message and returns a `MAACK` does not mean that the real EMX Hub will also accept the message. Nor is any checking made of the certificates used (see later), other than that the certificate presented by the client has been signed by the key that signed the EmuEMX's certificate.

### 7.12.4.5. Using the Demonstration Programs during your Development Phase

Provided that you take care to create valid EMX Messages, you can do most of your development and testing with the help of the demonstration examples. You will be developing either an Intermediary system

or a Fund Provider system. You can connect your system to the EMX Hub Emulator in place of the demo program of the same type and leave the demo program of the other type connected. That gives you a round trip messaging system which will help you in your development and testing.

You can run any of the example programs on different computer systems. If you do that, you will have to copy the Catalys Node installation to each system and you will need to modify the examples' configuration files, which are found in the `config` directory. You will have to change the "hostname" parameter of either of the client programs from "localhost" to whatever is appropriate for your configuration. You are free to change the port number also, though there is little need to do that.

If you are developing an Intermediary application, you will be able to connect it to EmuEMX and address test messages to CBFIX, which should also be connected to EmuEMX. The CBFIX DemoProviderClient will produce sensible (though trivial) response messages to any request and order messages that your application under development sends to it.

If you are developing a Fund Provider application you can certainly start by using the CAFIX demonstration program to generate some simple messages for you. But you will not get very far with that. However, the Catalys Node Server Demonstration Example (see above) can send messages of any complexity in a controlled manner. Simple copy and edit the data files in the `resources` directory, or create your own data files from scratch in Euroclear's defined EMX format, and you can use them as repeatable test cases to generate messages for sending to your program under development.

#### 7.12.4.5.1. Updating the demonstration system's properties files

The demonstration programs read their configuration details from a "properties" file. You can find the name of the properties file by reading the command file which executes each of the programs. Each "properties" file looks somewhat like this:

```
# Define the EMX Participant ID that we will connect with.
#
CompID=
<emphasis>ParticipantID</emphasis>
#
# Either connect to Trialling Hub...
#
#hostname=Test.FIXconnect.emx.co.uk
#hostport=4433
#
# ... or connect to the BasicHub EMX Emulator.
#
hostname=localhost
hostport=4433
#
# Specify the location of the keystore and truststore.
#
keystore=security/<emphasis>ParticipantID</emphasis>.keystore
truststore=security/<emphasis>ParticipantID</emphasis>.truststore
#
# Specify where we want log and persister files written.
#
```

```
LogBaseDir = .
Log4JConfig = config/<emphasis>ParticipantID</emphasis>.log4j.xml
#
# The next specification is needed only in the IFAClient.properties file.
# It specifies the Fund Provider to which our test messages will be sent.
#
defaultProviderID=
<emphasis>ProviderParticipantID</emphasis>
#
```

Lines starting with "#" are comments. You can use any text editor to make changes to the properties files. When you copy or move any of the demonstration programs you may well need to make at least some changes.

## 7.12.5. Developing Your System

### 7.12.5.1. Creating a Development Environment

You should start developing your own system in your own file area, of course, not in the Catalys Node directory structure. But, as mentioned previously, you may find that connecting to the examples and using the EMX Hub Emulator will be useful to you.

If you modify any of the examples, you should first copy them to your own file location, otherwise they could be overwritten if you upgrade your Catalys Node release. You should rename all of the files you copy to something that is meaningful to you. Within the copied files you will need to modify some of the path names that refer to the Catalys Node file system, such as the DTD files in the XML configuration files. You can use absolute or relative path names, as you prefer.

Some of the strategies you can use within which the demonstration examples will be of direct help to you are:

1. You can copy and extend the source code of either `DemoPrividerClient` or `DemoIFAClient`, which you will find in the `templates` directory. You can easily add additional EMX fields to the messages that these programs send and you can customise them to meet your needs. You will need a Java programmer to follow this route.
2. If you are developing an Intermediary system and you have a friendly relationship with a Fund Provider who uses EMX, you could ask the Fund Provider to connect his test system to your `EmuEMX` program. You will need to open your firewall to the relevant port, of course. Then you can send messages from your system under development to his test system and receive real response messages.
3. Likewise, if you are developing a Fund Provider system and you have a friendly relationship with an Intermediary who uses EMX, you could ask the Intermediary to generate messages for you by connecting his test system to your `EmuEMX`.

If you are asking another party to connect to your copy of `EmuEMX`, you will need to send him a copy of `EmuEMX.keystore` (if they are using Catalys Node) or `EmuEMX.pfx` (if they are using any other software).

He will need the keys and certificates contained in those files to communicate with your EMX Hub Emulator.

### 7.12.5.1.1. Generating Key Pairs and Keystore Management

Note: In 2016/2017 Euroclear announced changes in their Public Key Infrastructure introducing 2048-bit certificates and SHA256 digest algorithm. Catalys Node's EMX add-on fully supports these changes and customers only need to install 2048-bit certificates in order to make use of the upgraded security features. Full set of both 1024-bit and 2048-bit demonstration certificates is provided in security folder.

The Java keytool utility (a command line program that is supplied with Java, so you will have a copy on your system) can generate new key pairs. You do not have to do this, since you can use the keystore and truststore files that were provided with the EMX Integrator's demonstration examples. But, if you want to generate your own security keys, here is the command that we used to generate EmuEMX.keystore. Enter all of the arguments on the same line.

```
keytool -genkeypair -alias EmuEMX \
        -validity 10000 \
        -keyalg RSA \
        -sigalg SHA1withRSA \
        -keystore EmuEMX.keystore \
        -storepass password
```

Most of the required parameters are included in that command, and `keytool` will prompt you for the other things it needs. Enter your own personal and company details, and enter "yes" when asked to confirm that they are correct (or "no" if you need to correct anything). Then press return when asked for the key password, since the default is that it is the same as the store password.

This creates a Java JKS keystore file, which is perfect for use by the Catalys Node. But if you need to supply any other party with a Microsoft Personal Information Exchange file (PFX), you will need to copy the keystore to the PKCS#12 format used by PFX files. Use the keytool utility as follows:

```
keytool -v -importkeystore \
        -srckeystore ./EmuEMX.keystore \
        -srcstoretype JKS \
        -destkeystore ./EmuEMX.pfx \
        -deststoretype PKCS12
```

You can send this file to any other party, who will be able to use it in their own software. You will need to tell them that the password is "password".

If you need to provide another party with a certificate only, and not the private key as well, use `keytool` to create a PKCS#7 certificate file as follows:

```
keytool -export -file EmuEMX.cer \
        -alias EmuEMX \
        -keystore EmuEMX.keystore \
```

```
-storepass password
```

#### 7.12.5.1.2. Using `keytool` to view a keystore

The Java `keytool` utility can be used to list the contents of a keystore or a truststore. You need to specify the `-list` option and the `-v` (verbose output) option, and to specify the name of the keystore with the `-keystore` option. You can also specify the keystore's password on the command line, as illustrated below, but `keytool` will prompt you for the password if you don't. Copy the command below, changing the name of the keystore file to your own file name. Enter all of the arguments on the same line.

```
keytool -keystore CAFIX.keystore -storepass password -list -v
```

You will see full details of the key and/or certificate listed in the command window.

It is advisable to list the keystore and truststore that you are using, so that you can be certain that the contents of the keystore and truststore are what you believe you are using, and are not expired or otherwise invalid.

#### 7.12.5.1.3. Using `keytool` to view a certificate

The `keytool` utility can also display the contents of a certificate file. Copy the command below, changing the name of the keystore file to your own file name. Enter all of the arguments on the same line.

```
keytool -printcert -file certfile.cer
```

You will see full details of the certificate listed in the command window.

### 7.12.5.2. Connecting to the EMX Trialling System

Well in advance of the time when your system development has reached a stage where you are ready to connect to Euroclear's EMX Client Test System, you should apply to Euroclear to obtain a Participant ID and a certificate. Your Euroclear representative will agree a Participant ID with you and will give you the instructions for obtaining a certificate.

#### 7.12.5.2.1. Obtain an EMX Certificate for your Participant ID

The first part of this process can be carried out on any Windows computer, irrespective of whether or not you are running your Catalys Node EMX Integrator application on Windows.

Using Internet Explorer, go to Euroclear's [EMX web site](#). Select the "Digital Certificates" option, then select the link labelled "Trial certificate application form". When you have filled in the form, click on the "Submit Request" button. Your browser generates a private/public key pair at this stage, and sends the public key to Euroclear for inclusion in the certificate they will prepare for you. The private key remains in your browser, in a secure location. You should see a confirmation screen. Euroclear may take a day to deal with your request.

When Euroclear has prepared your certificates, you will receive an email containing a link to Euroclear's server from which you can download the certificate. Clicking the link will install the certificate into Internet Explorer. You must do this on the same computer from which you made your application.

#### 7.12.5.2.2. Make your Certificate available to the EMX Integrator

Your certificate and your private key will be in Internet Explorer's secure area. You need to export them to a file that you can make available for use by the EMX Integrator. To do this:

- Select `Tools` then `Internet Options` from the menu bar.
- Select the `Content` tab.
- Select `Certificates`.
- Highlight the certificate you want to export and press the `Export` button.
- A "wizard" will ask you some questions. Choose "Export the private key".
- Choose PKCS#12 and check "all certificates" and "strong encryption" if offered.
- Enter a password of "password" (or your own choice of password).
- Enter " `yourid.pfx` " as the filename for the exported file, where `yourid` is the Participant ID for this certificate. Specify that it should be stored in your installations's `security` directory (or a location of your choice).
- Press the `Finish` button and the operation will be carried out.

This will create a PFX file on the computer from which you applied for your certificate. You should copy this file to the computer on which you are running the Catalys Node, if different. Update your configuration file by changing the name of the `keystore` property's value to match the name and location of the PFX file you have exported, and change the name of the `truststore` property's value to access the `EMXRoot.cer` file supplied with the EMX Integrator, or a copy of it, or your own EMX Root certificate file if you wish to follow the next optional step.

*Note: In earlier versions of the Catalys Node you would have had to perform an additional step, namely the conversion of the PFX file into the JKS keystore format that the Catalys Node would require. This is no longer necessary, since the SSL Adapter and the EMX Integrator can now read both PFX files and CER files directly. You may still perform the conversion to JKS process if you wish. But there is really no benefit in doing so.*

#### 7.12.5.2.3. Optional: Download Euroclear's EMX Root Certificate

For your convenience, a copy of Euroclear's EMX Root Certificate is delivered with the Catalys Node EMX Integrator. You will find it in the `security` subdirectory of the examples you have previously run. You need to copy the `EMXRoot.cer` file (or the `EMXRoot.truststore` file if you prefer) to your own directory, or point your configuration to the supplied copy.

If you would like to download a fresh copy of Euroclear's EMX Root Certificate, or if you need to do so at any time in the future, the instructions are included here for completeness. The EMX Root Certificate

has a long validity period, but it will expire eventually and Euroclear will produce a new one which you will need to obtain in this way.

You can download Euroclear's EMX Root Certificate from Euroclear's Certificate Authority web site:

- Click on this link: [Install EMX Message System Root Certificate](#).
- Choose "Save" when asked to open or save.
- Browse in the file dialog to the location where you wish to save the certificate file. You may change the file name to "EMXRoot.cer" if you wish your name to match this documentation.
- When complete, you will be asked "Open", "Open folder" or "Close". You may choose whichever option you prefer. Note that, if you choose Open (which displays the certificate), there is no need to choose "Install", since this certificate only needs to be used with the Catalys Node and does not need to be installed in your browser.

You can now copy or move the EMX Root certificate to the `security` directory of your Catalys Node installation, or to any other location that you may have chosen to use. Ensure that your configuration file's `truststore` property's value refers to the new `EMXRoot.cer` file that you have downloaded.

Note: As mentioned above, in earlier versions of the Catalys Node you would have next had to create a JKS format truststore file from this certificate file. This is no longer necessary, since the SSL Adapter and the EMX Integrator can now read both PFX files and CER files directly. But if you do wish to create a JKS truststore file, simply run the command:

```
keytool -import \  
-file EMXRoot.cer \  
-alias EMX \  
-keystore EMXRoot.truststore \  
-storepass password
```

## 7.12.6. Performing the EMX Compliance Tests

Before Euroclear permits you to connect to the Live EMX Hub system, you must satisfactorily complete Euroclear's compliance tests. You should arrange a compliance test schedule with your Euroclear contact at a time when your system development and testing is nearly complete. You will be given various test scripts that you will be asked to run.

Most of the tests are concerned with "normal processing", where your Catalys Node client either remains connected to EMX throughout, or where you are asked to break the connection and reconnect without manually changing the FIX message sequence numbers. These tests are easily performed, since Catalys Node's normal behaviour is what is required.

Some of the tests, however, involve artificially adjusting one or other of the FIX message sequence numbers, in a way that Euroclear will specify to you. The purpose of these tests is to simulate a situation where one of the two FIX engines (either your Catalys Node client or the EMX Hub) has missed



some messages that the other party has previously sent. The FIX Protocol allows recovery of missed messages, and Euroclear wants to test that your FIX engine will indeed recover in the expected way.

The way you adjust the FIX sequence numbers depends on whether you are using the Catalys Node or whether you have built your own Java application built on the EMXClient toolkit.

Note that you cannot (or should not) reduce the outgoing sequence number or increase the expected incoming sequence number. Neither of these events are expected to be handled by a FIX engine, since they represent a fatal breach of the FIX Protocol.

### 7.12.6.1. Specifying sequence numbers when using the Catalys Node Server

The Catalys Node Server has a [Command Line](#) which gives you all of the options you need to make ad-hoc changes to the normal mode of operation. The Command Line is accessed by pressing enter in the terminal where you started the Catalys Node Server. Type "help" to get the list of available commands. Type "s\_list" to get a list of configured sessions. Your session should be able to be identified by `TargetCompID=EMX` and `SenderCompID=CAFIX`. You will need the ID of the session for all session commands. This appears on the left hand side of the Command Line output, #9 for example. First of all, type "s\_drop #9". This simulates a temporary loss of the network connection. the Catalys Node's normal behaviour is to try to reconnect straight away. You will see this occur within a few seconds. Your session carries on without any adverse impact.

However, this is not what you want when you are required to change the FIX sequence numbers. You need to prevent automatic reconnection, and you do that by first typing "s\_block #9" prior to "s\_drop #9". The "s\_block" command prevents automatic reconnection until you later select "s\_unblock".

So this time choose "s\_block #9" and then "s\_drop #9". The Catalys Node Server will sit idly until you unblock the connection. This gives you time to make any changes you need. Now choose "s\_out\_seq\_number #9 50", for example or any sequence number higher than the last sequence number that your FIX engine sent, which you can see on your console or log file output (outgoing sequence numbers are preceded with ">", incoming with "<"). Then choose "s\_unblock #9" and "s\_connect #9". Your FIX engine will reconnect but will be sending a higher sequence number than the other party expects. This will cause the other party (EMX or the EMX Emulator) to send a FIX `ResendRequest` message, which you will see on your console or log file, because it believes that it has missed receiving all of the messages in between the last sequence number that it received and the artificially high sequence number that it has now received. Your FIX engine will satisfy this `ResendRequest` by sending a `SequenceReset` message, which lets the other party know that it has not, in fact, missed any application messages.

You can also use the same procedure to change the next expected input sequence number. If you enter a number lower than the last sequence number that your FIX engine actually received, and then reconnect ("s\_unblock" and "s\_connect"), your FIX engine will receive a higher sequence number from the other party than it now expects and will believe that it has missed some messages from the other party. This results in your FIX engine sending a `ResendRequest` message, and the other party will resend

any application messages in the missing range, together with `SequenceReset` messages to fill any sequence numbers which have no application messages attached.

By using the "s\_block" and "s\_drop" commands you should be able to perform all of the abnormal processing tests that Euroclear ask for.

### 7.12.6.2. Specifying sequence numbers when using the EMXClient

You can program in your own control of sequence numbers, but you probably don't need to. If you want to break the current session and then restart it with a change to the sequence numbers you can simply kill your FIX client ( `Ctrl-C` in the Windows console, or `kill -9` in Unix, etc.).

If you then restart the EMXClient (or your application class that extends EMXClient), it will reconnect and carry on from where it had got to, which is exactly what you want when a temporary service disruption has occurred. The built-in persistence mechanism implements that behaviour by default.

But some of the Euroclear compliance tests require you manually to change the starting sequence numbers that the persistence mechanism would have used. To do that, before restarting your application, add one or both of these Java command line options to your startup script:

- " `-DoutMsgSeqNum= p` " : to set the first outgoing sequence number to `p` , and/or
- " `-DinMsgSeqNum= q` " : to set the first expected incoming sequence number to `q` .

These options only affect the first attempt at connecting to EMX.

## 7.12.7. Obtaining a Certificate for the Live EMX System

When you have successfully passed Euroclear's compliance tests, you will be allowed to apply for a certificate for use in connecting to the Live EMX System. The same EMX Root Certificate is used on all EMX sites, live and test, so you can simply copy your `EMXRoot.cer` to your live environment. But your live Participant ID will very likely be different from your test ID, and in any case you will need a certificate granting different permissions for connecting to the Live EMX System.

Go to the same link as given above under the heading [Obtain an EMX Certificate for your Participant ID](#), but this time select the link labelled "Digital certificate application form". Complete and submit the application form in the same way as before.

When Euroclear notifies you that your certificate is ready, follow exactly the same procedure to download and install the certificate in your live environment.

### 7.12.7.1. Renewing your Certificate

Your certificate will have an expiry date, probably one or two years after its issue date. When your certificate is nearing expiry, Euroclear will notify you and ask you to apply for a new one. You obtain this in exactly the same way as you obtain your original certificate.

Similarly, Euroclear will notify you when the EMX Root Certificate is nearing expiry. This has a longer validity date, so the need to renew it only occurs once every five to ten years. You can download a new EMX Root Certificate by following the instructions under the heading [Download Euroclear's EMX Root Certificate](#), and use it to replace your current `EMXRoot.cer`.

## 7.12.8. Other Information

### 7.12.8.1. Start & Stop Time

The EMX service is operational from 08:00 to 18:00 (UK time) on weekdays. The default behaviour of the EMX Integrator is to log out from EMX at 18:00 local time and sleep until 08:00 local time the next day (omitting Saturdays and Sundays).

If your EMX Integrator system is running on a computer that is NOT using UK time, you will probably want to change the default start and stop times.

You may also want to change the default start and stop times when you are developing and testing your system. There is no need to stop work at 18:00 — the EMX Emulator will be running 24/7, so why not carry on working!

Changing the start and stop times is easy, but the methods differ depending on whether you are using the Catalys Node Server or are building your own Java application by extending the EMXClient:

Catalys Node Server: You must edit the command file that starts the Catalys Node Server. Add the following statements to the java command line, directly after the space following "java" at the beginning of the line: " `-DEMXStartTime=08:40 -DEMXStopTime=23:00` ".

EMXClient: You can add two additional configuration lines to your "properties" file: " `StartTime = 08:40` ", and " `StopTime = 23:00` ".

### 7.12.8.2. Catalys Node Scheduler

If you are using the Catalys Node Scheduler, the default or custom scheduling of the EMX Integrator (as described in the previous section) is disabled. You must schedule the EMX Integrator using the Catalys Node Scheduler.

## 7.13. EMX Delimited File Adapter

### 7.13.1. Introduction

Clients can connect to the Catalys Node EMX Integrator using any of the connection technologies supported by CameronFIX. However, as a special convenience to our EMX clients, the EMX Integrator is supplied with an interface that mirrors the EMX web site's File Upload and Download capability. This interface is known as the EMX Delimited File Adapter.

You can choose whether to use the EMX Delimited File Adapter simply by configuring the Catalys Node Server either to use the EMX Delimited File Adapter or to use a different connection technology adapter. Some clients may wish to use the EMX Delimited File Adapter to start with and later migrate to a different connection technology.

This document describes how to configure the Catalys Node to use the EMX Delimited File Adapter and how the EMX Delimited File Adapter should be used.

### 7.13.2. Configuration and Initial Set Up

A Catalys Node Server configuration file that uses the EMX Delimited File Adapter is supplied as one of the [EMX Integrator example programs](#). Clients can copy and modify the configuration file to suit their own use. Any client who would like help with configuring their system is welcome to contact CameronTec FIX Support, who will be pleased to assist.

### 7.13.3. EMX Delimited File Adapter User Guide

Using the EMX Integrator with the Delimited File Adapter is very straightforward. The remainder of this document is a User Guide for the daily use of the Delimited File Adapter. You should begin by running the demonstration program referred to in the previous paragraph. After that, you should read this documentation, then copy and convert the configuration of the demonstration program to make it use your own EMX Participant IDs. You will then have a complete system running and ready to use.

#### 7.13.3.1. File Formats

The EMX Delimited File Adapter reads and writes Tab Separated Values (TSV) data files. The file formats are defined in template files that are part of the Catalys Node installation. The template files are located in the *template* subdirectory of the *templates/EMX* directory. The supplied file format templates match the file formats used by Euroclear's EMX web trading site, except for the Cash Settlement files (which are not available on Euroclear's EMX web site so have been defined by CameronTec). You will not need to amend these template files if you want to use Euroclear's standard file formats, but they can be customised if you wish.

**Note:** The `templates` directory of your Catalys Node installation contains demonstration examples and source code templates for you to copy. The file format templates are in the `templates/EMX/template` subdirectory of the Catalys Node installation.

The location of the file format templates is defined in the Catalys Node Server configuration file. You should copy the file format template files to a permanent location on your system. You should also copy and rename the Catalys Node Server configuration file, and update the configuration file with the location you have chosen for the file format templates. At the same time, you should choose proper permanent locations for all other Catalys Node files, such as the persistence and log files (see the Catalys Node Server documentaion).

If you do wish to modify the file format templates to suit your own needs, then it is easy to do so. Each row of a template file defines a column in the data file format. Rows representing columns that are never used can be removed, for example, so that those columns can be omitted from the data files you send and receive. And rows representing columns that always contain a constant value can have that value permanently set so that they also can be omitted from the data files. Anyone who wishes to modify the templates is welcome to contact CameronTec FIX Support for help and advice.

### 7.13.3.2. The XML Configuration File

The Catalys Node Server is configured via an XML file that is specified (on the command line, using the `-xmlconfig` directive) when the Catalys Node Server is started.

Most of the configuration is the same, whether or not the EMX Delimited File Adapter is being used. But the contents of the `SessionManager` section needs to follow this format if you are using the EMX Delimited File Adapter:

```
<SessionManager
  clientConnectionManager="com.camerontec.catalys.server.adapter.emx.manager.EMXConnectionManager">
  <!-- (see Note 1) -->
  <SourceMessageProcessors>
    <EMXMessageSigner id="EMXms">
      <!--(see Note 2) -->
      <SourceMessageThrottle
        id="smt"
        buffer="directory/MessageThrottleBuffer"
        messagesPerInterval="3"
        interval="1">
      <!--(see Note 3) -->
      <EMXDelimitedFileAdapter
        id="EMXsdfa"
        templateDirectory="directory/template"
        outgoingDirectory="directory/outgoing"
      />
      <!--(see Note 4) -->
    </SourceMessageThrottle>
  </EMXMessageSigner>
</SourceMessageProcessors>
<ListenerMessageProcessors>
  <EMXMessageVerifier id="EMXmv" verify="NONE">
    <!--(see Note 5) -->
    <EMXDelimitedFileAdapter
      id="EMXldfa"
      incomingDirectory="directory/incoming"
      incomingSwitchTimer="0"
    />
    <!--(see Note 6) -->
  </EMXMessageVerifier>
</ListenerMessageProcessors>
</SessionManager>
```

Notes:

1. This `SessionManager` configuration specifies that the connection manager to be used is the `EMXConnectionManager`. This is required for connecting to EMX.
2. You must specify the `EMXMessageSigner` as the outermost element of your `SourceMessageProcessors`. This message processor signs every message that you send to EMX using the key and certificate that has been defined in the surrounding `Session` element higher up in the configuration file. If you omit the `EMXMessageSigner` then your messages will not be signed and EMX will protest about this by dropping your connection!
3. This `SourceMessageThrottle` specification restricts the maximum rate of messages that the Catalys Node will send to EMX to three per second. EMX Co (as part of your contract with them) requires you to agree not to send more than three messages per second, so that no client takes an unfair share of the EMX Hub's resources at any time. Messages will be stored in a temporary buffer file if necessary, to keep within that limit. You need to specify a directory and file name for that file. We suggest you use `MessageThrottleBuffer` as the name, but you should choose a sensible directory location. A temporary storage area is fine.
4. You must specify the `EMXDelimitedFileAdapter` as the innermost element of your `SourceMessageProcessors`. This defines the source of the messages that are to be sent. You need to specify the directory where the file format templates can be found and you need to specify the directory where you will put files of data ready to be sent.
5. You should specify the `EMXMessageVerifier` as the outermost element of your `ListenerMessageProcessors`. If you set the `verify` parameter to `NONE`, as here, it will not in fact do anything. But you may later decide to verify incoming messages, and that can easily be achieved by changing this paramter to `ALL`, for example.
6. Although you have already specified the `EMXDelimitedFileAdapter` as the innermost element of your `SourceMessageProcessors`, you must also specify the `EMXDelimitedFileAdapter` as the innermost element of your `ListenerMessageProcessors`. This defines the target to which incoming messages will be sent. You do not need to specify the directory where the file format templates can be found, since you have already done so above (the specifications are shared). You need to specify the directory where the `EMXDelimitedFileAdapter` should put files of data messages received. You should also specify a value for the `incomingSwitchTimer` parameter. Zero (as here) means that incoming file sets will not be switched unless requested (see below under Incoming Message Files, Receiving a File). Otherwise you can specify a number of minutes after which the `EMXDelimitedFileAdapter` should close the incoming file sets and start writing new incoming data files.

See the [EMXDelimitedFileAdapter configuration reference](#) for a complete list of configuration attributes.

## 7.13.4. Daily Running

The Catalys Node EMX Integrator, once started, will run continuously. It will log on to the EMX Message System FIX Hub, using the Participant ID and certificate specified in the configuration file. The EMX Integrator will remain logged on to the Hub until the Hub sends a FIX Logout message at the end of the trading day, or will log out itself if that hasn't happened by 18:00. At that point the EMX Integrator will

sleep until 08:00 the following day (except Saturdays and Sundays). It then wakes up and logs in for a further day's trading. You thus never have to stop and restart the EMX Integrator (although you can choose to log out manually at any time if you wish).

If there is a break in the availability of the EMX Message System service, which happens rarely, the EMX Integrator will periodically attempt to make a reconnection. When the service is available again, it will log in and continue without any message loss. The ability to do this is part of the FIX protocol. It means that small service losses will very likely never be noticed by the client and no action is needed to perform a recovery. The ongoing support requirement is therefore minimal.

When the EMX Integrator is configured to use the EMX Delimited File Adapter, the configuration file must specify two directories, which may be local to the server or networked so that users can put files in and take files out from networked PCs. One directory is the `outgoing` directory, from where the EMX Integrator will take files of messages to be sent to the EMX Message System, the other is the `incoming` directory, which is where the EMX Integrator will put files of messages received from the EMX Message System.

In the simplest possible usage pattern, you need only to put a file of messages to be sent in the `outgoing` directory and the EMX Integrator will immediately read the file, convert each message to FIX and send it to the EMX Message System. A file can be sent by a simple "drag and drop", for example.

Messages received from the EMX Message System are converted from FIX to the delimited file format and placed in files in the `incoming` directory. The client can take files from here and pass them to whatever process is used to deal with incoming messages. A client who is already using EMXCo's file upload and download will already have processes in place to both produce and accept message files in this format.

The remainder of this document describes the outgoing and incoming process in full detail and explains how to achieve a higher or complete level of automation using the EMX Delimited File Adapter.

## 7.13.5. Outgoing Message Files

### 7.13.5.1. Naming Convention

The Delimited File Adapter needs to distinguish which type of messages is contained within any file placed in the `outgoing` directory. It does this by requiring that the first four characters of the file name are the message type. For example, a file containing orders must have a name that starts with "OINP" (the EMX Message System code for Order Input).

The file name must always end with ".tsv", which indicates that the file is a Tab Separated Values type.

The remainder of the file name can be anything the user desires. It is recommended to use some reference name or number, perhaps even a sequential number, so that the names are unique. The EMX Integrator does not require that the names are unique but if they are not the acknowledgement file (described below) of a previously used name may be overwritten when the name is used a second time.

The file name must therefore conform to this pattern:

*MsgType.Identifier.tsv*

Strictly speaking, the dot between MsgType and Identifier is not necessary. A space or any other character will serve. A dot makes a good visible separator, though, and will be used in this documentation.

For example, the following file names are acceptable:

- OINP.1.tsv,
- OINP.000253.tsv,
- TREQ\_Apr05.tsv,
- VREQ\_A1.tsv.

### 7.13.5.2. Sending a File

As soon as the EMX Integrator finds a file in its configured *outgoing* directory, it opens the file, reads each message, converts each message to FIX and sends it.

A file may contain any number of messages. There may be one message in the file, a few messages, or a thousand or more. The messages must all be of the same type, according to the file name as described above.

### 7.13.5.3. Validation and Error Handling

If there is an error in the file format itself, the EMX Integrator rejects the whole file and appends a ".error" extension to the file's name as an indication. This is unlikely to happen except in the early stages of using the EMX Integrator, since the file will most likely be produced by software to a consistently good format.

Assuming that the file format is valid, the EMX Integrator simply sends each message to the EMX Message System Hub. It makes no attempt to validate the contents of the message locally. The EMX Hub will validate every message it receives, and there is no need to pre-empt that process.

The EMX Hub sends back an acknowledgement ("MACK") message for each message received that passed the Hub's validation checks. It sends back an error acknowledgement ("MERR") for each message that failed the Hub's validation checks.

The EMX Integrator counts the number of acknowledgements received and will not begin sending any other file until all of the expected acknowledgements have been received.

The acknowledgements are written to two files: *MACK.ACKS.Identifier.tsv* and *MERR.ACKS.Identifier.tsv*, using the same Identifier as was used for the outgoing file.

Note, therefore, that you may want your identifier to be unique across all message types. The acknowledgement files are written to the *outgoing* directory.



If the file contained only valid messages, no "MERR" file will be produced. The absence of a "MERR" file is an easy check that all messages in the outgoing file have been accepted by the EMX Message System.

If the file contained only invalid messages, no "MACK" file will be produced. In any case, the number of MACKS and MERRS received will together equal the number of messages sent.

Please note this important difference between the EMX Message System's web-based File Upload and the EMX Integrator: Whereas the web-based File Upload will reject the whole file if it contains any error whatsoever, the Catalys Node EMX Integrator will always send every message to the EMX Message System (unless the file format is invalid). So a file of 120 messages of which 3 were rejected will nevertheless result in the 117 valid messages having been successfully sent. The three messages in error can be easily identified by processing the "MERR" file. These can be corrected and resubmitted via a new outgoing message file. You will not find good messages held up by being in the same file as bad ones.

While the EMX Integrator is writing the acknowledgement files, their file name will have an additional ".active" extension. Any file with a ".active" extension is still being processed by the EMX Integrator. When the EMX integrator has finished writing the file it removes the ".active" extension to indicate that the file can now be processed. At the same time, the EMX Integrator renames the original outgoing file with a ".done" extension. This indicates that it has been completely processed and that you can delete or archive it.

### 7.13.5.4. Simple Usage

Clients who already use the EMX Message System's web-based File Upload and Download service and who do not want to develop any additional software can use the EMX Integrator to send files as follows:

- Configure the EMX Integrator so that the `outgoing` directory is a network directory available to the users who currently use the File Upload service.
- When a file is ready for upload to the EMX Message System, the user now simply has to drag and drop the file into the `outgoing` directory. There is no need to log in or navigate through menus — the EMX Integrator is already logged in.
- Wait a short time for the MACK and/or MERR file to appear in the `outgoing` directory. Wait until their name ends ".tsv", indicating that they have been completely written, or (equivalently) wait until the original message file has been renamed with a ".done" extension.
- If there is no corresponding "MERR" file, that confirms that all of the messages in the outgoing file have been received and accepted by the EMX Message System. That is the equivalent of having the web site accept the whole file.
- If there is a "MERR" file, some of the messages have been rejected. The rejected messages and the reason for the rejection can be identified by looking at the "MERR" file. The messages in the "MERR" file are the same as the messages that the web site's File Upload would have produced. However, the valid messages will have been sent. So don't correct and resend the whole file or else you will send duplicate messages.

Any current user of the EMX Message System File Upload will gain immediate productivity benefits from switching to the Catalys Node EMX Integrator.

### 7.13.5.5. Advanced Usage

For those clients who are already producing message files in the EMX Message System's File Upload format, or are planning to do so, it is a simple matter to extend client's application software so that it places the outgoing message files directly into the EMX Integrator's `outgoing` directory and can read and process the returned `MACK` and `MERR` acknowledgements.

- Configure the EMX Integrator and the application software that produces the outgoing message files so that the software writes its outgoing files to the `outgoing` directory. It may be appropriate to use a secure local directory on the server, or it may be convenient to use a networked directory.
- When the application software creates a file of messages to send, the EMX Integrator immediately sends it.
- The application software should be programmed to wait until the original message file has been renamed with a `".done"` extension. This indicates that the EMX Message System has received and acknowledged all of the messages in the file. It should also check for a `".error"` extension and report any such occurrence.
- The application software should handle any errors that are reported in the corresponding `"MERR"` file. If there is no `"MERR"` file, all of the messages have been accepted by the EMX Message System.
- After handling any `MERRS`, corrected messages should be sent in a later file.
- After processing the acknowledgements, the `"MACK"`, `"MERR"` and `".done"` file should be deleted or archived.

An approach such as this allows outgoing messages to be handled with no human intervention.

## 7.13.6. Incoming Message Files

### 7.13.6.1. Naming Convention

The EMX Integrator names all incoming messages files in the following way:

*MsgType.Date.Time.tsv*

The *MsgType* indicates the message type contained in the file (and therefore the file format). The *Date* and *Time* are timestamps created when the first incoming message is received (after any previous message file was closed). The format of the *Date* identifier is `YYMMDD`. As well as giving an indication of when the file was started, it ensures that all file names are unique and that they sort into alphabetical order. The following file names are examples of file names that the EMX Integrator might create:

- `TBKD.140410.115033.tsv,`
- `ECNI.140410.131721.tsv,`

- `VRES.140410.150053.tsv`,
- `MERR.140410.115033.tsv`.

Note that all files that are created during the same time period will have the same time stamp, even though (say) the first `MERR` did not arrive until some time after the first `TBKD` arrived. Thus the time stamp is not guaranteed to represent the time that the first message in the file was received. It is the time that the first message in any of the files sharing the same timestamp was received.

Files that are presently being written by the EMX Integrator have a `".active"` extension. That extension is removed by the EMX Integrator when it has finished the file.

### 7.13.6.2. Receiving a File

When the EMX Integrator receives a message, if it already has a file open for messages of that type it simply appends the message to that file. If it doesn't already have a file for messages of that type it creates a new one.

Messages continue to be written to a single set of files, according to message type, until either of the following occurs:

- A configurable timer expires. The timer is set in the configuration file. If set to 30 minutes then exactly 30 minutes after the first message was received the current set of active files will be closed and made available to the client for processing.
- The client places a file called `"flag.switch"` in the `incoming` directory. When the EMX Integrator sees this file it immediately closes all of the current active files and makes them available to the client for processing.

Any messages arriving after one of those events will be written to a new file.

As with the acknowledgements of outgoing messages, while the EMX Integrator is writing the incoming message files their name will have an additional `".active"` extension. Any file with a `".active"` extension is still being processed by the EMX Integrator. When the EMX integrator has finished writing the file it removes the `".active"` extension to indicate that the file can now be processed.

Note that the `MERR` files that are written to the `incoming` directory are `MERRS` from a Product Provider, not from the EMX Message System. The file format is the same, however.

### 7.13.6.3. Simple Usage

Clients who already use the EMX Message System's web-based File Upload and Download service and who do not want to develop any additional software can use the EMX Integrator to receive files as follows:

- Configure the EMX Integrator so that the `incoming` directory is a network directory available to the users who currently use the File Download service.
- Configure the EMX Integrator so that received message files will be switched automatically every 30 minutes, 60 minutes, or at any choice of convenient time interval.

- Check for incoming message files throughout the day. When a file is found in the `incoming` directory and its extension is `".tsv"`, the file can be moved from the `incoming` directory and processed exactly as if it had been downloaded from the EMX Message System web site. Do not move any file that still has a `".active"` extension — wait until the EMX Integrator has finished writing it and made it a `".tsv"` file.

Any current user of the EMX Message System's web-based File Download will gain immediate productivity benefits from switching to the Catalys Node EMX Integrator.

### 7.13.6.4. Advanced Usage

For those clients who are already processing message files in the EMX Message System's File Download format, or are planning to do so, it is a simple matter to extend the application software so that it looks for its incoming message files in the EMX Integrator's `incoming` directory.

- Configure the EMX Integrator and the application software that processes the incoming message files so that the application software reads its incoming files from the `incoming` directory. It may be appropriate to use a secure local directory on the server, or it may be convenient to use a networked directory.
- Configure the EMX Integrator so that it does not switch incoming file sets according to a timer (set the timer value to zero or leave the setting entirely absent).
- Make the internal software create a `"flag.switch"` file in the `incoming` directory at regular intervals. It is possible to be a little sophisticated here, such as looking first to see if there are any `".active"` files, determining from their time stamp how long since the first message arrived, etc. Such sophistication is possible and can be useful, but is not necessary. The EMX Integrator changes the flag file's name to `"flag.ready"` when all active files have been closed and are ready for processing by the client's software.
- When there are new `".tsv"` files to process, process them using the application software. Do not process any `".active"` files.
- After processing each incoming `".tsv"` file, it should be deleted or archived.

An approach such as this allows outgoing messages to be handled with no human intervention.

## 7.13.7. Operational Issues

### 7.13.7.1. Hours of Operation

The EMX Integrator will connect to EMX between 08:00 and 18:00, Mondays to Fridays. At other times it will do nothing until the next weekday at 08:00. These times are appropriate for most clients but can be changed if necessary. The default times can be changed by specifying Java command-line parameters, eg `java -DEMXStartTime=08:30 -DEMXStopTime=17:00` etc.

Note that the EMX Integrator will connect to EMX on bank holidays. This causes no problems and avoids burdening clients with the maintenance of a list of non-operational days. Any client who wishes

to schedule connections to EMX with more complex requirements can purchase and use the Cameron Scheduler with the EMX Integrator.

### 7.13.7.2. Checking the Connection to EMX

Clients will want to know if their connection to EMX is not functioning during the expected hours of operation. If the EMX system has failed then urgent orders may need to be sent through other channels. If the EMX Integrator is not functioning as expected, an early warning of the fact can ensure prompt recovery.

There are two main checks that a client application should make to verify that the EMX Integrator is functioning as expected and that it is logged on to EMX:

- The EMX Integrator creates a file called "flag.connected" in the `outgoing` directory when it logs on to EMX and deletes the file when it logs out or the connection fails. If the "flag.connected" file is present, you can be confident that any message file placed in the `outgoing` directory will be sent immediately. Conversely, if the "flag.connected" file is not present at a time when it should be, the system's operators should be warned to investigate the problem.
- After a message file has been placed in the `outgoing` directory, the application that put it there should check that the corresponding `MACK` (or `MERR`) file arrives in the directory within a short time (two minutes should be ample). The EMX service level agreement guarantees that messages are acknowledged within 30 seconds, so if acknowledgements are not received reasonably promptly then there may be a fault with EMX and enquiries should be made in that direction.

It is also possible to parse the log file output for indications of abnormal behaviour. However, the two checks recommended above should be sufficient to ensure that you are made aware of any operational problems promptly.

## 7.14. C/C++ API

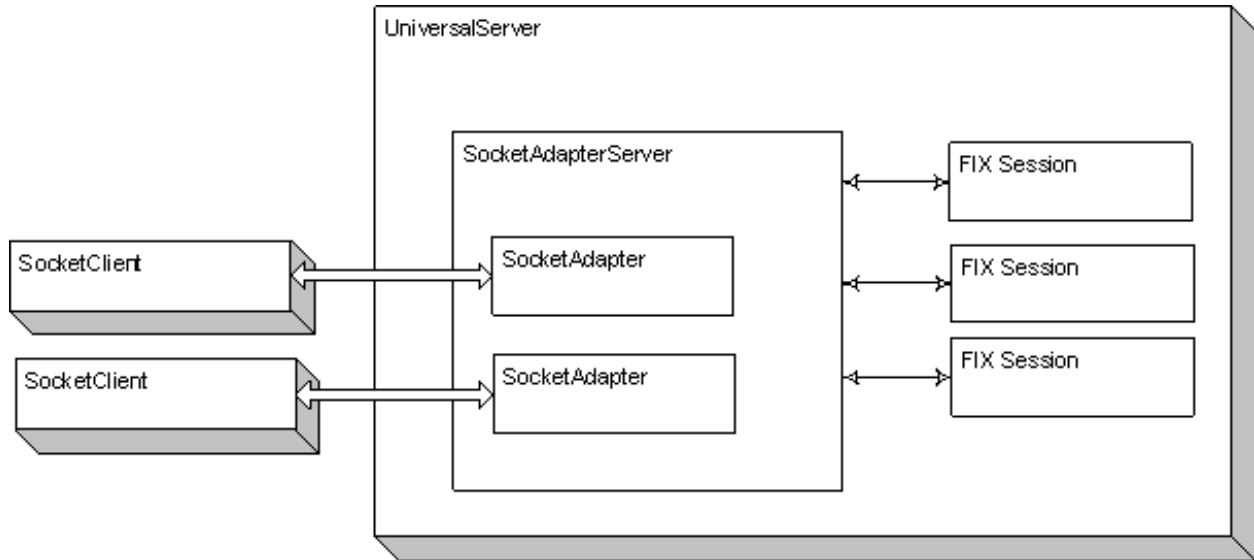
### 7.14.1. Introduction

The Catalys Node acts as a gateway process between your applications and your FIX counter-parties. The C/C++ API gives FIX access to your C or C++ application.

This API provides a static library to link into your application to connect to Catalys Node, which will in turn connect to FIX counter-parties. This API is designed for C and C++ users on Windows and Unix systems.

### 7.14.2. Overview

The following diagram shows how your application, labelled `SocketClient`, connects to the Node to communicate over one or more FIX sessions. A FIX session is a unique connection with a FIX counter-party.



Your application connects to the Node using a statically linked library. Use this library to establish a connection with the Catalys Node and pass FIX messages to and from your FIX counter-parties.

### 7.14.2.1. Data Transport

Behind the scenes, the C/C++ API will create a TCP socket connection to the Node, which will be configured to accept TCP socket connections by specifying the Socket Adapter in the Node's *config.xml*. There are two different Socket Adapter implementations available (Socket Adapter and Advanced Socket Adapter), having different characteristics with regard to server control and application topology:

- Number of applications that can connect to a single FIX session  
The Advanced Socket Adapter allows multiple client applications to connect to a single (or multiple) FIX sessions. Some clients use this to implement a GUI that can send and receive messages to any of the FIX counter-parties.
- Catalys Node Command Line Access  
In contrast, the simple Socket Adapter will only accept a single application connection per FIX session.
- Catalys Node Command Line Access  
With the Advanced Socket Adapter, you can give commands to the Node to start or stop sessions and set sequence numbers, among others.

### 7.14.3. Programming Notes

This API does no memory allocation using malloc and free (or other C++ allocation calls). Therefore, no memory pointers returned from any functions within the API require clean-up or `free()`.

### 7.14.4. Library

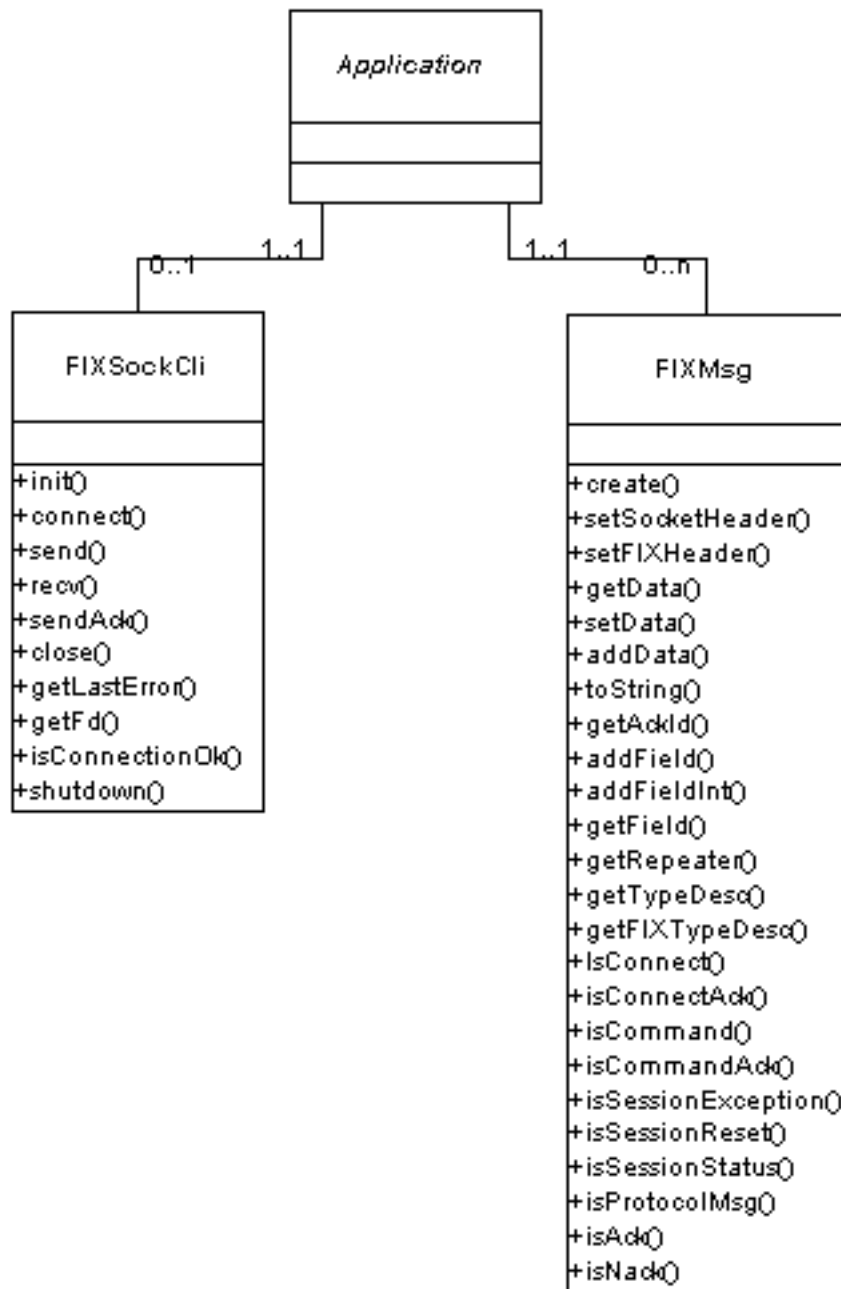
The C library has been tested with Microsoft Visual C++ 6.0 and GNU gcc 3.2.2.

A pre-built *sockapi.lib* and *simplapi.lib* are provided for Windows users under the `lib` directory. You can find the source code for the library under the *templates/src/cApi/c* directory.

To build the library for Linux or Solaris go to the templates directory, *templates/cApi/unix/Advanced* or *templates/cApi/unix/Simple*, and follow the instructions in *Readme.txt*.

## 7.14.5. Class Diagram

This diagram includes the two object classes for this library and an `Application` class as a place holder for the application since the library classes relate to the application rather than directly to one another.



### 7.14.5.1. Overview

From a high-level view, this API has a data transport class, `FIXSockCli`, and a data representation class, `FIXMsg`.

### 7.14.5.2. FIXSockCli

This class is used to transport data to and from the Socket Adapter. In order to use this class in both C and C++, the class has attributes only. The methods shown above are called by concatenating



class and method name and passing the object address as the first parameter. For example,  
`FIXSockCliInit(&connect).`

#### **7.14.5.2.1. Init() and Shutdown()**

These methods will initialize the object and, for MS Windows, load and unload the socket DLL.

#### **7.14.5.2.2. Connect()**

`Connect()` opens a socket connection and handshakes with the Node (with `CONNECT` messages). While `Init()` must be called prior to `Connect()`, a side effect of `Connect()` under Windows is to reload the winsock DLL if it has been unloaded resultant of a socket error.

`Connect()` must also be called after `Send()` or `Recv()` return an error. Errors reported by these functions will result in the automatic closure of the socket.

One important aspect of `Connect()` is the recovery mechanism. In case of a disconnect, a socket reconnect must be followed by another `CONNECT` message. If the data field is empty, every message from the beginning of the session (typically the start of the trading day) will be resent from the Socket Adapter. To receive only messages missed while disconnected, the connect message must contain a table of sessions (since Socket Adapter and the API can accommodate simultaneous separate FIX sessions) along with the last `AckId` received for the session.

The recommended approach is to let the API maintain this persistence automatically by calling `SetPersist()` after `Init()`.

#### **7.14.5.2.3. Send() and Recv()**

These methods will handle I/O, but may return errors requiring a another call to `Connect()`. `Send()` will send the Socket Adapter buffer held in `FIXMsg` which may contain any of the defined Advanced Socket Adpater message types.

`Recv()` will wait for messages from the Socket Adapter and populate a `FIXMsg` object. Any Socket Adapter message type may be received, but of primary importance are FIX messages, ACK, and NACK. Receipt of a FIX message will require an Ack reply (using `SendAck()`).

#### **7.14.5.2.4. SendAck()**

Send acknowledgment Socket Adapter message for a given FIX message.

#### **7.14.5.2.5. GetFd()**

Provides access to the file handle for direct system calls (e.g., setting socket attributes or for an event manager).

#### **7.14.5.2.6. SetPersist()**

This function activates the libraries built-in inbound message-id persistence and auto-reconnect ability. This is tied into the `Recv()` and `SendAck()` functions. When a FIX message is `SendAck()`'d, the id is saved in an internal array and backed up to a flat file. When the application is restarted and

`SetPersist()` is called, the internal array is rebuilt from the flat file. When `Connect()` is called (on start-up or after a socket disconnect), the connect protocol includes the last message id received for each session.

#### 7.14.5.2.7. ResetPersist()

Clears internal and flat file list of sessions and last message IDs received for those sessions.

### 7.14.5.3. FIXMsg

`FIXMsg` will be the logical class name for this set of methods to manipulate message content. Actual function names are the concatenated class and function. E.g., `Create()` actual name is `FIXMsgCreate(FIXMsg*)`.

This class hold contains an envelope for the message (used for communication with the Node) as well as a FIX message within. Messages other than a FIX message (such as a message ACK) do not contain a FIX message in the envelope (in the `data` field). It is possible to use this data field to import non-FIX style messages into the Node to be converted to FIX from within your custom Java class.

#### 7.14.5.3.1. Create()

`Create()` will initialize the user provided structure (so no memory allocation required). It is assumed that the message allocation will take place on the stack (auto variable, such as `FIXMsg x;` declared in the first lines of a function), so it will be kept small for efficiency.

#### 7.14.5.3.2. SetSocketHeader()

This is used to construct non-FIX messages, but rather session control messages, and will not normally be used by the application programmer. Set the Socket Adapter message header for this message's message envelope containing message length, message type (e.g., `CONNECT`, `PROTOCOL_MSG`), and message data. This function would typically be used to construct `CONNECT` or `COMMAND` messages, rather than FIX messages.

#### 7.14.5.3.3. SetFIXHeader()

To send a FIX message, call `Create()` and `SetFixHeader()` to build the Socket Adapter message envelope with the FIX header put in the data field. Subsequent `AddField()` calls will append fields to the FIX header.

#### 7.14.5.3.4. AddField() and GetField()

These functions will operate on the given message using the defined tag IDs. This will keep the class size down and still provide compile time checking. This does not, however, provide type safety.

#### 7.14.5.3.5. GetAckId()

Returns the Socket Adapter's ASC ID message field. This is required to acknowledge receipt of FIX messages, typically with `FIXSockCliSendAck()`.

#### 7.14.5.3.6. ToString()

Return the FIX output buffer of the FIX message in its current state. This will be encapsulated in the Advanced Socket Adapter's small message header and trailer.

#### 7.14.5.3.7. AddData(), SetData(), and GetData()

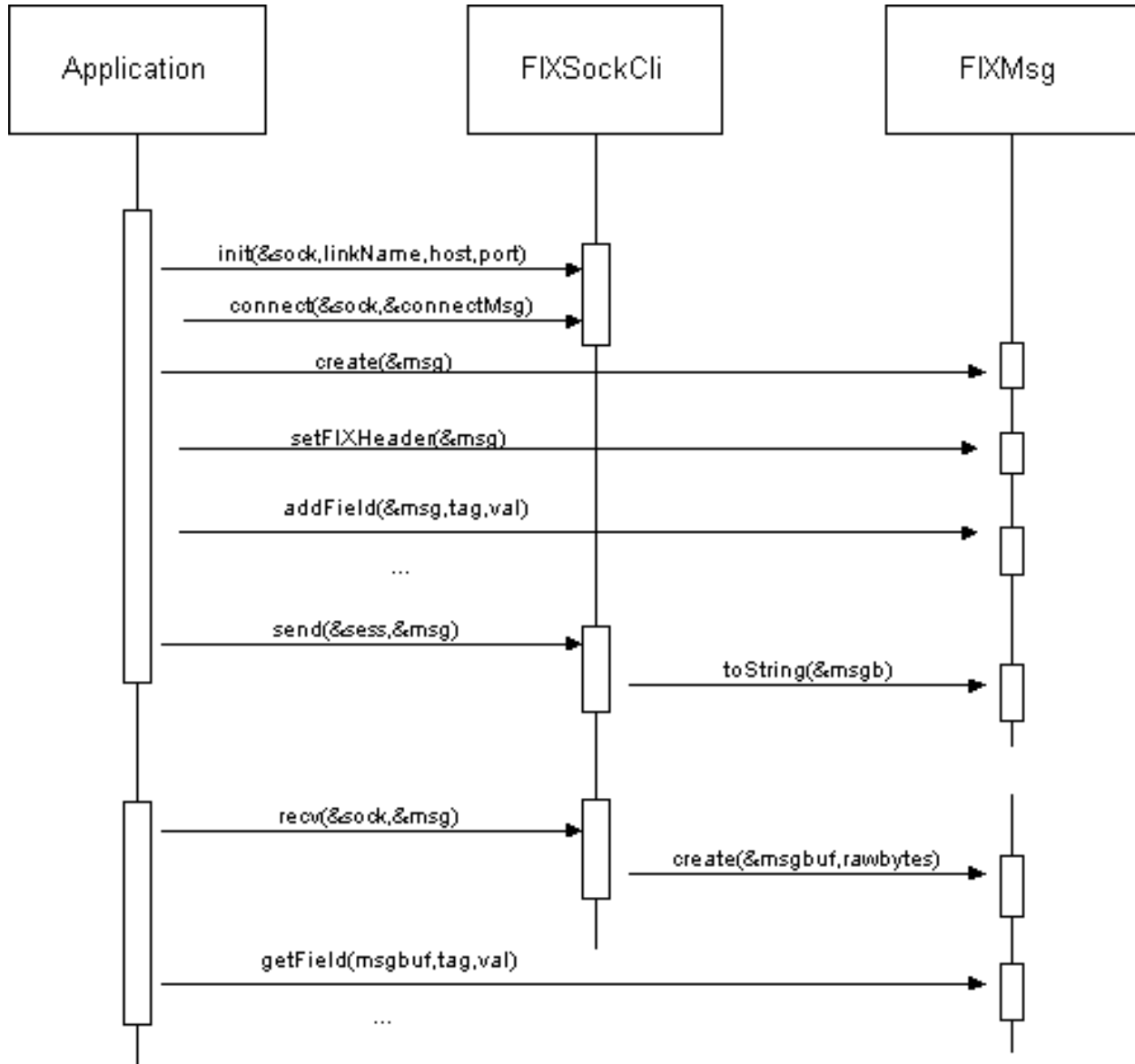
These functions manipulate the Socket Adapter's message data field. For `PROTOCOL_MSG`, this is typically where FIX messages are written with `AddField()`, but can also contain a user object that the Node can convert to FIX.

These functions return non-zero if the buffer will not fit.

`GetData()` returns a pointer and length to the field. `SetData()` copies given buffer to the field, discarding any data already in that field. `AddData()` appends given buffer to any existing data in the message's data field.

### 7.14.6. Sequence Diagram

This diagram shows the sequence of events to create a session and send a message to the Node. The `Recv()` should repeat as many times as needed until an ACK is received. `Recv()` may also return a number of other message types while waiting for ACK. E.g., incoming FIX messages may be received while waiting for the ACK for an outbound FIX messages.



### 7.14.7. Templates

Working templates are located in *templates/cApi*. The full source code for the cApi template is available in *templates/src/cApi/cpp*.

### 7.14.8. Sample user code

The full source code for the SocketClient Template is available at *templates/src/cpp/main.cpp*.

An excerpt is listed here.

```
FIXSockCli conn;
```

```

main () {
    /* Open a socket connection to the Advanced Socket Adapter */
    FIXSockCliInit(&conn, "fix_link1", "localhost", 7200);

    /* Set inbound seq# persistence */
    FIXSockCliSetPersist(&conn, "fix_link1_persist.dat");

    while (1) {
        if (FIXSockCliConnect(&conn, NULL)) {
            printf("Socket connect error: %d\n", FIXSockCliGetLastError(&conn));
        }

        /* Create a NewOrderSingle and send it */
        {
            FIXMsg msg;
            FIXMsgCreate(&msg);
            FIXMsgSetFIXHeader(&msg, "ack id",
                FIXMSG_NEWORDERSINGLE, "myCompID", "counterpartyID", NULL, NULL, NULL, NULL);
            FIXMsgAddField(&msg, FTAG_CLORDID, "ABC");
            FIXSockCliSend(&conn, &msg);
        }

        /* Wait for the Ack message */
        while (1) {
            FIXMsg inMsg;
            if ( FIXSockCliRecv(&conn, &inMsg) ) {
                /* report socket error */
                break;
            }

            /* process ack or nack */
            if ( FIXMsgIsAck(&inMsg) ) {
                /* process ack back to internal application or DB */
                break;
            }
            else {
                /* Handle incoming message */
            }
        }
    }

    /* Clean up connection */
    FIXSockCliShutdown(&sock);
}

```

## 7.15. Microsoft .NET API

### 7.15.1. Introduction

The .NET API enables .NET-based systems to connect directly to the Node. It uses a lightweight object hierarchy for creating and transforming FIX messages and a lightweight protocol for interfacing with the Node without any dependencies on COM or other third-party interoperability solutions.

The Node has a simple .NET API available, which is fully managed code, and can be used in any language that is a first-class citizen in the Microsoft Common Language Infrastructure.

The current implementation performs the Java-to-.NET interoperability using a socket layer, and a byte-array representation of the message, and as such, the .NET client need not be on the same machine as the Node running the Socket Adapter to which the .NET adapter connects.

## 7.15.2. Installation

The .NET API requires that the [Socket Adapter](#) be configured.

The machine running the .NET client must have the .NET 2.0 or newer runtime installed.

The supplied template requires that you have the Microsoft .NET SDK 2.0 (or later) installed to operate. If you wish to look at the template source code, you will need to have Microsoft Visual Studio 2008 or newer installed.

## 7.15.3. Getting Started

Refer to the template in *templates/DotNetAdapter/windows*. We include a sample program, *AdapterClientExample.exe*, that demonstrates how to use the .NET API.

On the machine that is running the FIX engine:

1. Start the sell-side server by running *sell.bat* or *sell.sh*
2. Start the buy-side server by running *buy.bat* or *buy.sh*

On the machine that will be hosting the .NET client:

1. Launch the correct *AdapterClientExample.exe* executable based on your Windows version.
2. Check this program's console for output showing the client connecting to the Node as well as sending and receiving messages and events.

The .NET client is now established to the buy side and receiving all incoming messages and events.

## 7.15.4. The Catalys Node .NET API

The sample console application demonstrates the basic operation of the API, which is described in more detail here. What follows is a step-by-step guide to implementing a .NET FIX client.

### AdapterProperties

Create an instance of `AdapterProperties` object with the appropriate connection details and settings to the underlying Socket Adapter. This object is used to initialise the `AdapterClient` class, which is the main class for interacting with the Node.

Properties	Notes
ARG_HOSTNAME	<p>The IP or hostname of the host running the Node.</p> <p>This field is required.</p>
ARG_PORT	<p>The port the SocketAdapter is listening on.</p> <p>This field is required.</p>
ARG_USE_ACKS	<p>Boolean setting to determine if the .NET client should await acknowledgement from the Node for sent messages and also send acknowledgements for received messages. Should a message not be acknowledged within time specified by ARG_ACK_TIMEOUT, the client will issue an <code>ExceptionEvent</code> to your exception callback, allowing you to handle the event and recover the message that was not acked.</p> <p>The default value is <code>false</code>.</p>
ARG_ACK_TIMEOUT	<p>The time in milliseconds that the Client will wait for acknowledgement from the Node.</p> <p>The default value is 10000 ms (10 secs).</p>
ARG_USE_LARGE_SIZE	<p>Boolean setting that determines whether or not large message sizes should be used in transmission to and from the Socket Adapter. This should be set to <code>true</code> if messages will be sent/received that are greater in size than 999999 bytes. The corresponding attribute <code>largeMessages</code> should also be set on the Socket Adapter to match this setting.</p> <p>The default value is <code>false</code>.</p>
ARG_LOGFILE	<p>The logfile which will be used for logging informational messages. If this attribute is not set then the log information will be written to the console. The log file can either be in the current directory (e.g. <code>logfile.txt</code>) or a full path and file name (e.g. <code>c:\Vogfile.txt</code>).</p>

Properties	Notes
ARG_BUFFERING_OFF	<p>Boolean setting to enable or disable the TCP NoDelay flag.</p> <p>The default value is <i>false</i>, i.e. buffering is on.</p>
ARG_PING_INTERVAL	<p>The time in milliseconds between pings during periods of inactivity. The client will ping the Node, and if it does not respond within 100ms, the client will disconnect.</p> <p>The default value is 10000 ms (10 secs). To disable, set to 0.</p>

```
// Instantiate an AdapterProperties object and set the properties
AdapterProperties props = new AdapterProperties();
props[AdapterProperties.ARG_HOSTNAME] = Host.Text;
props[AdapterProperties.ARG_PORT] = int.Parse(Port.Text);
props[AdapterProperties.ARG_USE_ACKS] = true;
props[AdapterProperties.ARG_USE_LARGE_SIZE] = true;
props[AdapterProperties.ARG_LOGFILE] = LogName.Text;
props[AdapterProperties.ARG_PING_INTERVAL] = 3000;
```

## IAdapterClient

The `IAdapterClient` interface defines the interface that should be implemented by all .NET clients to the Node. The current implementation provided uses sockets to communicate between the FIX engine and .NET. This reduces the dependency on any external software needed for ease of integration. To initialise an `IAdapterClient` with an `AdapterProperties` use:

```
IAdapterClient client = new SocketClient(props);
```

## Event Handling

The `IAdapterClient` interface defines three .NET callbacks for handling the arrival of messages, exceptions, connection status and acknowledgments from the Node. The events are called `MessageEventHandler`, `ExceptionHandler` and `ConnectionEventHandler`. They require methods with a specific signature. To implement these events, create the following methods with the correct signatures within your class:

```
private void IncomingException(object sender, ExceptionEventArgs args)
{
    this.logger.error("Exception received from adapter",args.RaisedException);
}
```



## Technology Adapters

```
private void IncomingMessage(object sender, MessageEventArgs args)
{
    this.logger.info("Message received: " + args.FixMessage.ToString());
}

static void ConnectionEvent(object sender, ConnectionEventArgs args)
{
    this.Logger.info("Connection Event " + args.Status);
}
```

If you are also using the attribute `fullMessageAcks` in the Socket Adapter, in order to receive an acknowledgement when a message is sent out over FIX, you should also use the method:

```
private void IncomingAck(object sender, MessageEventArgs args)
{
    this.logger.info("Acknowledgement received. Message sent.");
}
```

And then register the methods with the appropriate handlers once you have created the client:

```
// Register the IncomingMessage method to the list of observers
client.MessageEvent += new MessageEventHandler(IncomingMessage);
client.ExceptionEvent += new ExceptionEventHandler(IncomingException);
client.ConnectionEvent += new ConnectionEventHandler(ConnectionEvent);
//if required
client.MessageAckEvent += new MessageEventHandler(IncomingAck);
```

## Sending and Receiving Messages

Once you have initialised a client, you may start receiving messages (and possibly exceptions) by issuing a `Connect()` call to the FIX engine. Call `Disconnect()` to stop receiving messages and sever the link to the FIX engine. The `Send(Message msg)` call is used to send messages.

```
client.Connect();
// You should start receiving messages into your callback method now
// Create a message using the Message class.
// The Constants class contains a definition of every Field and Message
// type defined in the FIX protocol specification.
FIX.Message message = new FIX.Message();
message.SetField(Constants.TAGMsgType, Constants.MSGOrder);
message.SetField(Constants.TAGOrderQty, "1000");
message.SetField(Constants.TAGPrice, "12.34");
message.SetField(Constants.TAGSide, Constants.SIDE_Buy);
message.SetField(Constants.TAGSymbol, "CAM");

// Send the message
client.SendMessage(message);

// Request Session Status
```

```
client.getSessionStatus();
```

When sending a message, you can choose to receive acknowledgments from the Node when the message is sent out over FIX (i.e. when it has finished all pre-processing, including the addition of the sequence number). To do this, you can set the attribute `fullMessageAcks` on the Socket Adapter and add an event listener for the acknowledgement as described in the preceding section.

## Handling and Transforming messages

The callback methods described earlier get called whenever a message arrives. A `Message` object is a lightweight .NET object that represents the fields of the message, but you may wish to convert it to another format. The `ITransformer` interface is designed to accommodate a broad range of message formats and convert them into and from this internal format that is used throughout the .NET API. All message data fields are transformed by an `ITransformer` implementation. The .NET API provides an implementation called the `RawTransformer`, however you may wish to provide your own implementation to convert to a proprietary format. The `RawTransformer` takes raw FIX messages in the form of byte arrays and converts them into `Message` objects and vice versa, from which you may create strings, MSMQ messages, write to `DataSources` etc.

To convert a FIX message within a callback event handler:

```
private void IncomingMessage(object sender, MessageEventArgs args)
{
    ITransformer transformer = new RawTransformer();

    // Get the message from the event args
    FIX.Message fixMessage = args.FixMessage;

    // Iterate through a list of fields
    List<Field> fields = fixMessage.Fields;
    for(Field f in fields) {
        this.logger.info(f.Name + " = " + f.Value);
    }

    // Transform the message to a byte array
    byte[] msgBytes = (byte[])transformer.toObject(fixMessage);

    // DefaultEncoding defined elsewhere to be UTF8
    String msgStr = SocketClient.DefaultEncoding.GetString(msgBytes);
    this.logger.info("Message received: " + msgStr);
}
```

### 7.15.5. Catalys Node Configuration

The .NET API interacts with the Node via the Socket Adapter, which reduces its dependency on external software requirements for Java/.NET interoperability. The following is an example configuration:

```
<?xml version="1.0"?>
```

```

<!DOCTYPE Config SYSTEM "CatalysConfig.dtd">
<Config>
  <Application>
    <Sessions>
      <Session fixversion="4.2"
        heartbeat="60"
        counterpartycompid="Bigbroker"
        compid="FatsFast Market">
        <Connections>
          <SocketConnection id="sc" listenport="2000"/>
        </Connections>
        <SessionManager>
          <SourceMessageProcessors>
            <SocketAdapter id="dotNet_socket_adapter"
              port="7201"
              ackTimeout="1000"
              alwaysUseMsgId="true"/>
          </SourceMessageProcessors>
          <ListenerMessageProcessors>
            <MessageProcessorReference id="dotNet_listener"
              refid="dotNet_socket_adapter"/>
          </ListenerMessageProcessors>
        </SessionManager>
      </Session>
    </Sessions>
  </Application>
</Config>

```

The above XML configuration file defines the `SocketAdapter` as a message processor. The `port` attribute defines the listening port to which the .NET client application will connect.

## Socket Adapter Attributes

The main [Socket Adapter](#) attributes to be considered when using in conjunction with .NET API are shown below.



### Note

The .NET API requires the `alwaysUseMsgId` attribute to be set to `true`.

Attribute	Notes
<code>ackTimeout</code>	The Socket Adapter can be configured to wait for acknowledgments from the client. When configured to do so, when the server sends a <code>SocketMessage.RECV_MESSAGE</code> , the client should respond with a <code>SocketMessage.SEND_ACK</code> as

Attribute	Notes
	<p>soon as it is done processing the message. The <code>ackTimeout</code> indicates the amount of time (in milliseconds) the server should wait for the acknowledgment before it considers the connection dead and disconnects.</p> <p>The default is "0", indicating the server should not wait for any acknowledgments.</p> <p>Optional.</p>
port	<p>This is the port on which the Socket Adapter should listen for a client connection.</p> <p>The default port is "5005".</p>
alwaysUseMsgId	<p>This is a <code>boolean</code> attribute available for backward compatibility. If true, acknowledgments from the socket client are expected to just include the message id from the original message. Otherwise the body of the original message is expected. Default is <code>false</code>, but this must be set to <code>true</code> when using the .NET API.</p>
largeMessages	<p>This is a <code>boolean</code> attribute available for backward compatibility. If true, the <code>MessageLength</code> field will be 9 bytes. If false it will be 6 bytes. This needs to be set to true to allow messages of size greater than 999999 bytes to be transmitted. Default is <code>false</code>.</p> <p>Optional. If this is set, the corresponding property <code>AdapterProperties.ARG_USE_LARGE_SIZE</code> should also be set.</p>

## Implementation Note

If your Socket Adapter and .NET client are configured to use acknowledgments and your code sends messages upon receiving another message, e.g. it replies with `ExecutionReport` on receiving `NewOrderSingle` message, then you should decouple sending by using separate thread to send a message reply.

# Chapter 8. Programming Guide

## Table of Contents

8.1. Custom Components .....	331
8.1.1. Component Lifecycle .....	331
8.1.2. Application Lifecycle .....	334
8.1.3. Overriding the Lifecycle Methods .....	335
8.1.4. Custom Message Processors .....	339
8.1.5. Custom Selectors .....	344
8.1.6. Custom Services .....	346
8.1.7. Custom Monitors .....	348
8.1.8. Accessing Global Configuration and Runtime .....	350
8.1.9. JMX Component Operations .....	351
8.2. CLI Commands .....	354
8.2.1. Introduction .....	354
8.2.2. Implementation .....	354
8.3. Tasks .....	356
8.3.1. Introduction .....	356
8.3.2. TaskBase Implementation .....	356
8.3.3. PartyTaskBase Implementation .....	357
8.4. Persistent Objects .....	358
8.4.1. Introduction .....	358
8.4.2. Collection Registry .....	358
8.4.3. Compacting a Collection .....	360
8.4.4. Removing a Collection .....	360
8.4.5. Collections .....	360
8.4.6. Externalizers .....	362
8.5. Message Factories .....	364
8.5.1. Introduction .....	364
8.5.2. Implementation .....	364
8.6. Message Transformers .....	365
8.6.1. Introduction .....	365
8.6.2. Implementation .....	365
8.6.3. Configuration .....	366
8.7. Repeating Groups .....	366
8.7.1. Introduction .....	367
8.7.2. Adding a Repeating Group .....	367
8.7.3. Parsing Repeating Groups .....	367
8.8. JMX API Advanced Operations .....	368
8.8.1. Configuring via JMX .....	368

8.8.2. Monitoring via JMX .....	370
8.9. Authentication .....	371
8.9.1. Introduction .....	371
8.9.2. References .....	371
8.9.3. FIX Authentication Implementation .....	372
8.9.4. FIX Authentication Configuration .....	372
8.9.5. Login Module Configuration .....	372
8.9.6. Support for Custom Login Module Implementations .....	372
8.9.7. FIX Authentication Templates .....	373

## 8.1. Custom Components

### Introduction

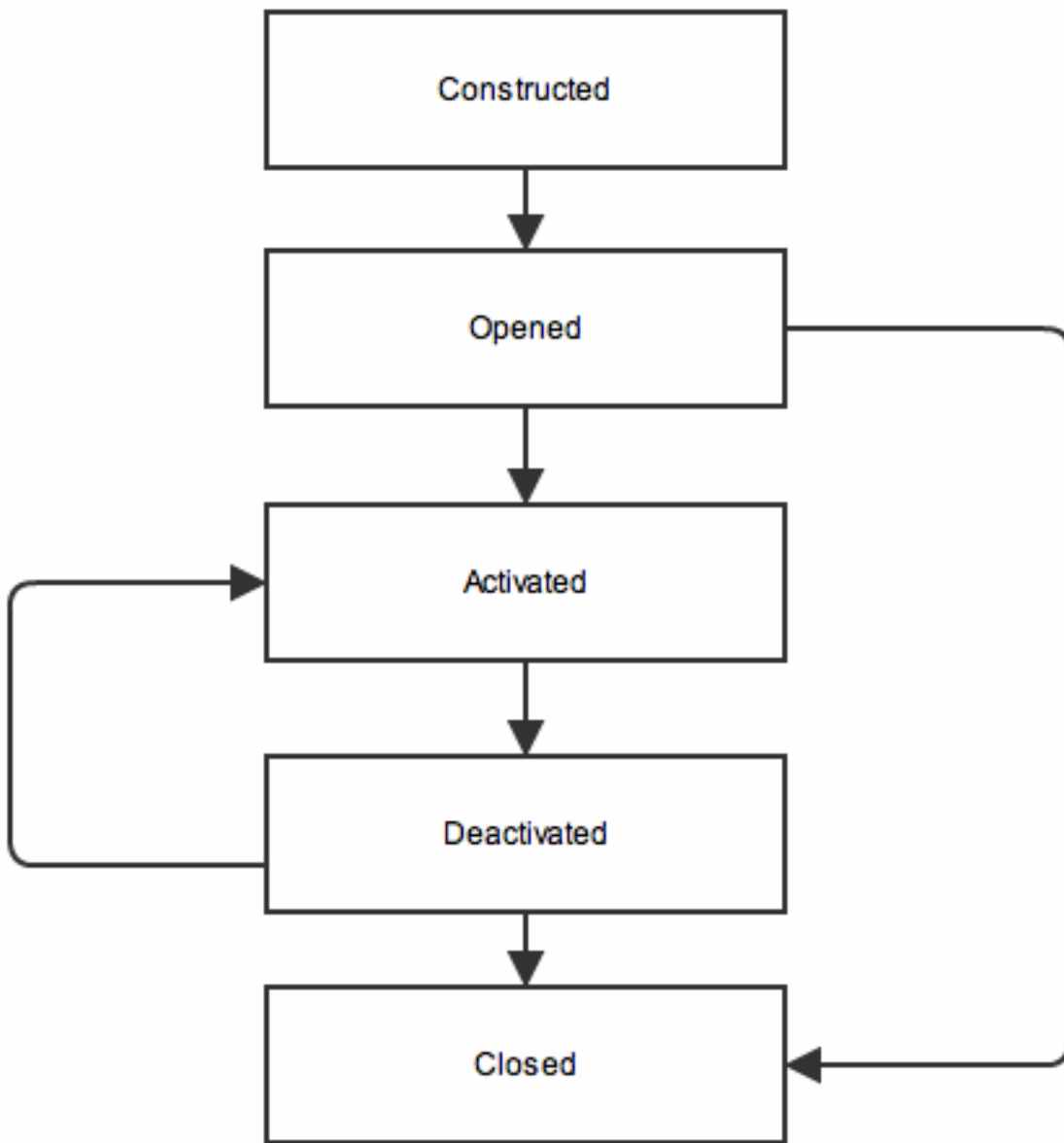
Many of the Node's features are implemented as *components*, which include message processors, services, monitors and selectors. Users can create their own components in order to implement specific functionality or business logic that is not provided by the built-in components.

From a programmer's perspective, a component is anything that implements the [IComponent](#) interface. This interface has a well-defined lifecycle to which an implementation must adhere, ensuring that the component is always in the correct internal state as the Node itself changes state. The interface also provides methods to access the built-in configuration subsystem.

We recommend implementing the `IComponent` interface by extending the [ComponentHelper](#) class. It provides default implementations of the interface's methods, allowing you to override only the methods that need to be customized.

#### 8.1.1. Component Lifecycle

All components follow this lifecycle:



State	IComponent Method	What an Implementation Should Do
Constructed	The component object is created via its default constructor.	<ul style="list-style-type: none"> <li>Custom components must provide a public no-argument constructor or a default constructor. See <a href="#">this section</a> below if your custom component needs to access the global configuration or runtime APIs.</li> </ul>

State	IComponent Method	What an Implementation Should Do
		<ul style="list-style-type: none"> <li>• The constructor should do as little as possible and should not have side effects.</li> <li>• The component should not depend on components being constructed in a certain order.</li> </ul>
Opened	<a href="#">openComponent()</a> is called. This method is only called once during the lifecycle.	<ul style="list-style-type: none"> <li>• Configure itself as needed according to its attributes.</li> <li>• Allocate any resources that will remain open during activation/deactivation state changes.</li> <li>• The <a href="#">ComponentHelper implementation</a> marks the component as open and registers itself with the component registry. It also calls the <a href="#">refreshStateFromAttributes()</a> method, which is where the component's configuration attributes should be parsed and applied.</li> </ul>
Activated	<a href="#">setComponentActive(true)</a> is called. This method may be called multiple times during a component's lifecycle, including successive calls with the same value.	<ul style="list-style-type: none"> <li>• If running in HA mode, allocate any resources that should only be open on the primary Node.</li> <li>• Begin processing and be ready to accept messages and events.</li> <li>• The <a href="#">ComponentHelper implementation</a> records that the component is active.</li> </ul>
Deactivated	<a href="#">setComponentActive(false)</a> is called.	<ul style="list-style-type: none"> <li>• Stop processing and stop accepting messages and events.</li> <li>• Release any resources that were allocated in <code>setComponentActive(true)</code>.</li> </ul>



State	IComponent Method	What an Implementation Should Do
		<ul style="list-style-type: none"> <li>The <a href="#">ComponentHelper implementation</a> records that the component is deactivated.</li> </ul>
Closed	<a href="#">closeComponent()</a> is called. No further IComponent methods will be called.	<ul style="list-style-type: none"> <li>Release any resources that were allocated in <code>openComponent()</code>.</li> <li>The <a href="#">ComponentHelper implementation</a> records that the component is closed and removes it from the component registry.</li> </ul>

## 8.1.2. Application Lifecycle

The transition of components from state to state is triggered by changes in the Node's lifecycle. The Node transitions between these three states:

- STOPPED: The Node is not running and no components are open or active.
- STARTED: The Node is running and all components are active (unless manually deactivated).
- STAND\_BY: The Node is a secondary HA node and all components are opened. Only permanent services are active.

### Application Startup

When a standalone Node starts up, it transitions from STOPPED to STARTED after:

1. All components are constructed
2. `openComponent()` is called on each component
3. Components are activated by calling `setComponentActive(true)`

See the [state transition table](#) for details about the order in which components are opened and activated. Note that components configured under a specific session are managed by their respective `SessionManager`, which calls the lifecycle methods on its child components. For example, when `SessionManager.openComponent()` is called, it will in turn call `openComponent()` on each of its message processors, selectors, etc.

### High Availability Mode

When starting in HA mode, a Node first transitions from STOPPED to STAND\_BY after:

1. All components are constructed

2. `openComponent()` is called on each component
3. Only permanent services are activated. A permanent service is one that should always be activated at startup, regardless of whether the Node is in a primary or secondary role. See [IComponent.SERVICE\\_ALWAYS\\_ACTIVE](#) for details on how to implement this.

At this stage all components other than permanent services are still deactivated.

If the Node then becomes primary — either at startup or after failover — the application transitions from `STAND_BY` to `STARTED`, and all components are activated via `setComponentActive(true)`, except for permanent services which will have already been activated.

When a primary node becomes a secondary during failover, it transitions from `STARTED` to `STAND_BY` and components other than permanent services are deactivated via `setComponentActive(false)`.

## Application Shutdown

When the application shuts down (e.g. via the `exit` CLI command), it transitions from `STAND_BY` or `STARTED` to `STOPPED` after:

1. If the application is in the `STARTED` state, components are deactivated via `setComponentActive(false)`.
2. If the application is in the `STAND_BY` state, any permanent services are deactivated via `setComponentActive(false)`.
3. `closeComponent()` is then called on all components. The implementation should not depend on components being closed in a certain order.

### 8.1.3. Overriding the Lifecycle Methods

For simple components, the default lifecycle management provided by `ComponentHelper` may be sufficient. But if your component requires configuration or needs to allocate additional resources before activating, you will need to override one or more of the lifecycle methods.



#### Important

When overriding any lifecycle methods in `ComponentHelper`, be sure to call the corresponding `super` method as well.

## Configuring a Component

It is likely that your component will need one or more configuration attributes. These can be specified in the XML configuration by nesting a `Properties` element beneath your component:

```
<Session counterpartycompid="CLIENT" compid="BROKER" fixversion="4.4" heartbeat="60">
  <Connections>
    <SocketConnection id="sc" listenport="2003"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors/>
    <ListenerMessageProcessors>
      <GenericProcessor class="com.example.CustomProcessor" id="customProcessor">
        <Properties id="props">
          <Property name="customAttribute" value="someValue"/>
        </Properties>
      </GenericProcessor>
    </ListenerMessageProcessors>
  </SessionManager>
</Session>
```

To read the configuration attributes at runtime, override the `refreshStateFromAttributes()` method. This method is automatically called when a component is opened and when it is safe for a component to reconfigure itself according to its attributes. From here, attributes can be retrieved using the `getAttribute(String)` method. We also provide a helper class, [AttributeUtility](#), which simplifies this process and provides type-specific attribute retrieval with optional default values.

```
package com.example;

import com.camerontec.catalys.server.component.ComponentHelper;
import com.camerontec.catalys.util.attributesupport.AttributeUtility;

public class CustomComponent extends ComponentHelper
{
    private volatile boolean customAttribute;

    @Override
    public void refreshStateFromAttributes() throws Exception
    {
        super.refreshStateFromAttributes();

        // get the attribute as a boolean
        final Object attr = getAttribute("customAttribute");
        customAttribute = attr == null ? false : Boolean.valueOf(attr.toString());

        // or do the same thing using AttributeUtility
        customAttribute = AttributeUtility.getBooleanAttribute("customAttribute", this, false);
    }
}
```



## Important

Care should be taken to provide appropriate synchronization and/or visibility for configuration data that is accessed by the component's runtime processing, as configuration and processing occur in different threads.

Components are configured when they are opened, but configuration changes can also occur after startup. A developer must decide whether each custom attribute is *read-only* (cannot be changed after startup) or *refreshable* (can be changed after startup).

All read-only attributes should be applied from `openComponent()`, which is only called once during the lifecycle. All refreshable attributes should be applied from `refreshStateFromAttributes()`.

## Opening, Activating and Closing a Component

If your component needs any other resources before activating, you should allocate them from `openComponent()` and deallocate them from `closeComponent()`.

After the component has been opened and configured, it will then be activated via `setComponentActive(boolean)`. This method should be overridden if you need to perform any other operations before your component can process messages, for example starting a new thread, opening a socket, connecting to a database, etc. Conversely, when the Node deactivates your component, it will call this method again; in this case you should close or deallocate any such resources. The `boolean` argument indicates whether the Node is trying to activate or deactivate your component.



## Note

When using your component in High-Availability installations, keep in mind that `openComponent()` is called on the primary and all secondary servers. However, `setComponentActive(true)` will be called only on the server that is currently the primary. Therefore any resources you wish to allocate only on the primary server should be allocated from `setComponentActive(true)`.

Here is a simple implementation of these methods:

```
package com.example;

import com.camerontec.catalys.server.component.ComponentHelper;
```

```
public class CustomComponent extends ComponentHelper
{
    private volatile SomeResource someResource;

    @Override
    public boolean openComponent() throws Exception
    {
        final boolean wasOpen = super.openComponent();

        if (!wasOpen)
        {
            // allocate necessary resources
            someResource = new SomeResource();
        }

        return wasOpen;
    }

    @Override
    public boolean closeComponent() throws Exception
    {
        final boolean wasOpen = super.openComponent();

        if (wasOpen)
        {
            // release resources allocated in openComponent()
            someResource.close();
        }

        return wasOpen;
    }

    @Override
    public boolean setComponentActive(boolean active) throws Exception
    {
        final boolean wasActive = super.setComponentActive(active);

        if (!wasActive && active)
        {
            someResource.start();
        }
        else if (wasActive && !active)
        {
            someResource.stop();
        }

        return wasActive;
    }
}
```

## 8.1.4. Custom Message Processors

### Introduction

A message processor is a component that processes FIX messages. Message processors are configured under a session and can be chained together. They can process messages in the inbound or outbound direction, or even both directions.

The following example shows the most basic message processor. It simply prints out each inbound FIX message it receives.

```
package com.example;

import com.camerontec.catalys.core.message.IFIXMessage;
import com.camerontec.catalys.server.processor.FIXListenerBase;
import com.camerontec.catalys.util.messaging.MessageEvent;

public class BasicProcessor extends FIXListenerBase
{
    @Override
    public void onMessageFromFix(MessageEvent event, IFIXMessage message)
    {
        System.out.println(message);
    }
}
```

To use this processor, a `GenericProcessor` element is configured in a session's message processing chain:

```
<Session counterpartycompid="CLIENT" compid="BROKER" fixversion="4.4" heartbeat="60">
  <Connections>
    <SocketConnection id="sc" listenport="2003"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors/>
    <ListenerMessageProcessors>
      <GenericProcessor class="com.example.BasicProcessor" id="basicProcessor"/>
    </ListenerMessageProcessors>
  </SessionManager>
</Session>
```

### Message Processor Class Implementation

Your custom processor must implement a specific interface depending on which direction messages will flow through it. We provide abstract base classes that you should extend in order to implement the interface:

Message Direction	Interface	Base Class
Inbound	<a href="#">IFIXListener</a>	<a href="#">FIXListenerBase</a>
Outbound	<a href="#">IFIXSource</a>	<a href="#">FIXSourceBase</a>
Inbound and Outbound	<a href="#">IFIXSource</a> and <a href="#">IFIXListener</a>	<a href="#">FIXSourceListenerBase</a>

These base classes extend `ComponentHelper`.

## Processing Inbound Messages

If your component will be processing inbound messages, you should extend either `FIXListenerBase` or `FIXSourceListenerBase`.

Messages will be given to your processor via the `onMessageFromFix` method. This is where you can read, enrich, validate or perform any other processing on the FIX message before it is handed off to the next message processor in the chain. If you throw an `Exception` from this method, a standard FIX reject message will be constructed for you automatically using the text of the exception.

Two other methods, `onConnectionStatus` and `onReset`, can be overridden if you wish to be notified of session events. Please refer to the [Javadoc](#) for a complete explanation of each method in the `IFIXListener` interface.

```
package com.example;

import com.camerontec.catalys.core.message.IFIXMessage;
import com.camerontec.catalys.server.processor.FIXListenerBase;
import com.camerontec.catalys.util.messaging.MessageEvent;

public class BasicProcessor extends FIXListenerBase
{
    @Override
    public void onMessageFromFix(MessageEvent event, IFIXMessage message)
    {
        // add a custom tag to the inbound message
        message.addField(6000, "some text");
    }

    @Override
    public void onConnectionStatus(MessageEvent event, boolean connected)
    {
        System.out.println("The session is " + (connected ? "connected" : "disconnected"));
    }

    @Override
    public void onReset(MessageEvent event)
    {
        System.out.println("The session was reset");
    }
}
```

```
}
```

The message processor must be configured in the session's listener message processing chain:

```
<Session counterpartycompid="CLIENT" compid="BROKER" fixversion="4.4" heartbeat="60">
  <Connections>
    <SocketConnection id="sc" listenport="2003"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors/>
    <ListenerMessageProcessors>
      <GenericProcessor class="com.example.BasicProcessor" id="basicProcessor"/>
    </ListenerMessageProcessors>
  </SessionManager>
</Session>
```

## Processing Outbound Messages

If your component will be processing outbound messages, you should extend either `FIXSourceBase` or `FIXSourceListenerBase`.

When a message originates earlier in the message processing chain, it will pass through your message processor via the `onMessageToFix` method. This is where you can read, enrich, validate or perform any other processing on the FIX message before it is handed off to the next message processor in the chain and ultimately sent out the FIX session.

```
package com.example;

import com.camerontec.catalys.core.message.IFIXMessage;
import com.camerontec.catalys.server.processor.FIXSourceBase;
import com.camerontec.catalys.util.messaging.MessageEvent;

public class BasicOutboundProcessor extends FIXSourceBase
{
    @Override
    public void onMessageToFix(MessageEvent event, IFIXMessage message)
    {
        // add a custom tag to the outbound message
        message.addField(6000, "some text");
    }
}
```

If your message processor is creating new messages to send out the FIX session, send them via the `fireMessage(IFIXMessage)` method. They will pass through any other configured message processors under which your processor is nested; they will not pass through any message processors that are nested beneath.

```
private void sendMessage() throws Exception
{
```



```
final IFIXMessage message = getToFIXMessageFactory().createFIXMessage("D");
message.addField(11, "Order1");
message.addField(55, "IBM");
fireMessage(message);
}
```

An outbound message processor must always be configured in the session's source message processing chain:

```
<Session counterpartycompid="CLIENT" compid="BROKER" fixversion="4.4" heartbeat="60">
  <Connections>
    <SocketConnection id="sc" listenport="2003"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors>
      <GenericProcessor class="com.example.BasicOutboundProcessor" id="basicOutboundProcessor"/>
    </SourceMessageProcessors>
    <ListenerMessageProcessors/>
  </SessionManager>
</Session>
```

## IFIXMessage Reuse

Messages are delivered to processors as `IFIXMessage` objects. For efficiency, the Node will try to reuse an `IFIXMessage` instance whenever possible so as not to create garbage.

If your message processor does not retain references to the `IFIXMessage` objects passed via `onMessageFromFix` and `onMessageToFix`, it is good practice to indicate that it's safe to reuse message objects that have passed through your processor. This is accomplished by calling `setStoresMessages(boolean)` from your constructor or from `openComponent()`.

```
package com.example;

import com.camerontec.catalys.server.processor.FIXListenerBase;

public class BasicProcessor extends FIXListenerBase
{
  public BasicProcessor()
  {
    setStoresMessages(false);
  }
}
```

More details about message reuse are available in the [Performance Tuning](#) chapter.

## Auto Forwarding

By default, after a message processor has processed a message, it automatically forwards the message to the next processor in the chain. In some cases, however, it may be desirable to prevent a message

from being forwarded. For example, the `CatalysRulesEngine` message processor disables auto-forwarding so that it can prevent dropped messages from being delivered upstream.

To disable auto-forwarding in your message processor, call [setAutoForwarding\(boolean\)](#) from the constructor or from `openComponent()`. Setting this to `false` means that no message will be forwarded beyond your processor unless you explicitly do so. To forward a message manually, call [forwardEvent\(MessageEvent\)](#).



### Important

When auto-forwarding is disabled, all events including messages, exceptions, resets and connection status events will not be auto-forwarded. Be sure to handle all of these events by overriding the corresponding methods and manually forwarding as needed. See the example below.

The following example illustrates a processor that drops certain messages and forwards all other events:

```
package com.example;

import com.camerontec.catalys.core.message.IFIXMessage;
import com.camerontec.catalys.server.processor.FIXListenerBase;
import com.camerontec.catalys.util.messaging.MessageEvent;

public class CustomProcessor extends FIXListenerBase
{
    public CustomProcessor()
    {
        setAutoForwarding(false);
    }

    @Override
    public void onMessageFromFix(MessageEvent event, IFIXMessage message) throws Exception
    {
        if (message.hasField(6000))
        {
            System.out.println("dropping this message");
        }
        else
        {
            forwardEvent(event);
        }
    }

    @Override
    public void onConnectionStatus(MessageEvent event, boolean connected) throws Exception
    {
        forwardEvent(event);
    }
}
```

```
@Override
public void onException(MessageEvent event, Exception exception) throws Exception
{
    forwardEvent(event);
}

@Override
public void onReset(MessageEvent event) throws Exception
{
    forwardEvent(event);
}
}
```

## JMX Operations

A custom processor will appear in the Dashboard or any JMX client without any extra effort. It will be available as a `Configuration.Application.MessageProcessor` MBean with a `com.camerontec.-catalys.server.management.mbean.CustomMessageProcessorConfigurationMBean` configuration class. This class exposes attributes via the `customAttributes` attribute and the `setCustomAttribute` operation.

See [JMX Component Operations](#) for more details about managing custom components via JMX.

## More Examples

The source code for two of our built-in message processors, `SampleMessageSource` and `SampleMessageListener`, is included in the `templates/src/com/camerontec/catalys/server/test/` directory of the Node installation.

## 8.1.5. Custom Selectors

### Introduction

A selector is a simple component which when passed an object decides whether or not that object is "selected", based on some internal criteria of the selector. For details on how and when to use them, refer to [Selectors](#) in the configuration chapter.

Two selectors, [MsgTypeSelector](#) and [SelectByTagValue](#), are bundled with the Node. Refer to the `templates/Selectors` directory for configuration examples; the source code can be found in `templates/src/com/camerontec/catalys/server/messageselector`.

### Selector Class Implementation

Selectors must implement the [ISelector](#) interface. Since they are components, we recommend using [ComponentHelper](#) as a base class and overriding the [lifecycle](#) methods as necessary.

The `ISelector` interface has just one method: `isSelected(Object)`. In the following example, messages with symbol "IBM" are selected.

```
package com.example;

import com.camerontec.catalys.core.message.IFIXMessage;
import com.camerontec.catalys.server.component.ComponentHelper;
import com.camerontec.catalys.util.messaging.IMessage;
import com.camerontec.catalys.util.messaging.MessageEvent;
import com.camerontec.catalys.util.notification.ISelector;

public class BasicSelector extends ComponentHelper implements ISelector
{
    @Override
    public boolean isSelected(Object obj)
    {
        if (obj instanceof MessageEvent)
        {
            final IMessage message = ((MessageEvent)obj).getMessage();

            if (message instanceof IFIXMessage)
            {
                final IFIXMessage fixMessage = (IFIXMessage)message;
                return fixMessage.getValueAsString(55, "").equals("IBM");
            }
        }

        return false;
    }
}
```

## Using the Selector

A selector is configured in the listener message processing chain of a session using the `GenericSelector` element. Alternatively, you can [define your own element](#) in the DTD.

```
<Session counterpartycompid="CLIENT" compid="BROKER" fixversion="4.4" heartbeat="60">
  <Connections>
    <SocketConnection id="sc" listenport="2002"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors/>
    <ListenerMessageProcessors>
      <GenericSelector class="com.example.BasicSelector" id="selector">
        <!-- nested processor here -->
      </GenericSelector>
      <!-- default processor here -->
    </ListenerMessageProcessors>
  </SessionManager>
</Session>
```

## JMX Operations

A custom selector will appear in the Dashboard or any JMX client without any extra effort. It will be available as a `Configuration.Application.Selector MBean` with a `com.camerontec.-catalys.server.management.mbean.CustomSelectorConfigurationMBean` configuration class. This class exposes attributes via the `customAttributes` attribute and the `setCustomAttribute` operation.

See [JMX Component Operations](#) for more details about managing custom components via JMX.

## 8.1.6. Custom Services

### Introduction

A service is a component that provides functionality which is typically shared by one or more message processors (or even by other services). For more details on how and when to use a service, please refer to [Services](#) in the configuration chapter.

For a working example, see *templates/CustomService* and the corresponding source code file *templates/src/com/camerontec/catalys/server/test/SampleService.java*.

### Service Lifecycle

Since services are available to all message processors, the component lifecycle guarantees that they are:

- Opened before all message processors
- Activated before all messages processors
- Deactivated after all messages processors
- Closed after all message processors

### Service Class Implementation

Services must implement the [IService](#) interface. Since they are components, we recommend using [ComponentHelper](#) as a base class and overriding the [lifecycle](#) methods as necessary.

The following example shows a simple service with a method that prints a string to standard out:

```
package com.example;

import com.camerontec.catalys.server.component.ComponentHelper;
import com.camerontec.catalys.util.service.IService;

public class CustomService extends ComponentHelper implements IService
{
    private volatile String serviceName;

    @Override
    public String getServiceName()
```

```
{
    return serviceName;
}

@Override
public void setServiceName(String serviceName)
{
    this.serviceName = serviceName;
}

public void helloWorld()
{
    System.out.println("hello world");
}
}
```

## Using the Service

To deploy the service, add a `GenericService` element to the Node's configuration:

```
<Application id="app">
  <Services>
    <GenericService class="com.example.CustomService" id="customService"/>
  </Services>
  <Sessions>
    ...
  </Sessions>
</Application>
```

All configured services are registered with the global [ServiceRegistry](#).

To actually use the service, for example from a message processor, it must be obtained from the registry and cast to the expected type. From there, you can call any one of its public methods.

```
package com.example;

import com.camerontec.catalys.core.message.IFIXMessage;
import com.camerontec.catalys.server.processor.FIXListenerBase;
import com.camerontec.catalys.util.messaging.MessageEvent;
import com.camerontec.catalys.util.service.ServiceRegistry;

public class CustomProcessor extends FIXListenerBase
{
    private volatile CustomService service;

    @Override
    public boolean openComponent() throws Exception
    {
        final boolean wasOpen = super.openComponent();

        if (!wasOpen)
        {
            service = (CustomService)ServiceRegistry.getInstance().getService("customService");
        }
    }
}
```

```

    }

    return wasOpen;
}

@Override
public void onMessageFromFix(MessageEvent event, IFIXMessage message) throws Exception
{
    service.helloWorld();
}
}

```

## Custom Configuration with HA

Services are not limited to any particular functionality, and as such they are not required to use the Node's standard configuration subsystem. If your service employs its own configuration mechanism and the Node is running in High Availability mode, then you may wish to replicate your configuration across the cluster. This can be achieved by implementing the [IServiceWithReplicatedConfiguration](#) interface instead of [IService](#).

## JMX Operations

A custom service will appear in the Dashboard or any JMX client without any extra effort. It will be available as a `Configuration.Application.Service` MBean with a `com.camerontec.-catalys.server.management.mbean.CustomServiceConfigurationMBean` configuration class. This class exposes attributes via the `customAttributes` attribute and the `setCustomAttribute` operation.

See [JMX Component Operations](#) for more details about managing custom components via JMX.

## 8.1.7. Custom Monitors

### Introduction

A monitor is an application-level component that listens to events from all configured sessions. It can also be used to customize session behavior.

### Monitor Class Implementation

To listen to session-level events only, the monitor should implement the [ISessionStatusEventListener](#) interface.

To listen to session-level events and session manager-level events, the monitor should implement the [ISessionManagerListener](#) interface.

The interfaces have one key method, `onEvent(CatalysEventObject event)`, via which sessions and session managers deliver events. The `CatalysEventObject` argument contains an integer property indicating the event type; these types correspond to a constant defined in [ICatalysEventType](#).

Since a monitor is a component, we recommend using [ComponentHelper](#) as a base class and overriding the [lifecycle](#) methods as necessary.

The following example shows a simple monitor that prints each event:

```
package com.example;

import com.camerontec.catalys.server.component.ComponentHelper;
import com.camerontec.catalys.server.sessionmanager.ISessionManagerListener;
import com.camerontec.catalys.util.notification.CatalysEventObject;

public class SimpleMonitor extends ComponentHelper implements ISessionManagerListener
{
    @Override
    public void onEvent(CatalysEventObject event)
    {
        System.out.println("Received event " + event.getType() + " from " + event.getSource());
    }

    @Override
    public boolean shouldReceiveHighVolumeEvents()
    {
        return false;
    }
}
```

In order to add the monitor to the Node configuration, define a `GenericMonitor` element:

```
<Application id="app">
  <CustomMonitors>
    <GenericMonitor class="com.example.SimpleMonitor" id="simpleMonitor"/>
  </CustomMonitors>
  <Sessions>
    ...
  </Sessions>
</Application>
```

We also provide an abstract base class, [SessionStatusListenerBase](#), to use as an alternative to manually implementing the interfaces mentioned above. This class inspects each event and fires a specific method corresponding to each event type. You can then override only the methods for the events you wish to handle.

A working demonstration of a custom monitor is included in *templates/CustomMonitor*. The source code for the monitor used in the template can be found at *templates/src/com/camerontec/catalys/server/test/SampleMonitor.java*.

## Customizing Session Behavior

A session customizer can be used to change the default behavior for: the Logon message exchange, if a counterparty should be accepted and if a message should be resent during a resend request.



To override the default session-level behavior, the custom monitor should implement the [ISessionCustomizer](#) interface. Refer to the [Javadoc](#) for details on all possible customizations.

We provide an abstract base class, [SessionCustomizerBase](#), which implements `ISessionCustomizer` and allows you to override only those methods you wish to customize.

An example of a session customizer is provided at *templates/src/com/camerontec/catalys/server/test/SampleMonitorCustomizer.java*.

## JMX Operations

A custom monitor will appear in the Dashboard or any JMX client without any extra effort. It will be available as a `Configuration.Application.CustomMonitor` MBean with a `com.camerontec.-catalys.server.management.mbean.CustomMonitorConfigurationMBean` configuration class. This class exposes attributes via the `customAttributes` attribute and the `setCustomAttribute` operation.

See [JMX Component Operations](#) for more details about managing custom components via JMX.

## 8.1.8. Accessing Global Configuration and Runtime

Custom components requiring access to the global configuration or runtime APIs can do so by declaring a constructor with the required interfaces as parameters. This is often useful if you need to reference other components' runtime configurations. The available APIs include:

- [ConfigurationAgent](#)
- [ConfigurationObjectRegistry](#)
- [ConfiguredObjectRegistry](#)
- [IConnectionPointConfiguration](#)

When the component is created, a constructor will be called with the correct instances passed as arguments. For example:

```
public MyComponent(ConfigurationAgent configurationAgent)
{
    this.configurationAgent = configurationAgent;
}
```

Or:

```
public MyComponent(ConfigurationAgent agent,
                   ConfigurationObjectRegistry config,
                   ConfiguredObjectRegistry runtime,
                   IConnectionPointConfiguration conn_points)
{
    this.agent = agent;
    this.config = config;
}
```

```
this.runtime = runtime;
this.conn_points = conn_points;
}
```

The configuration may then be accessed via one of these interfaces. For example:

```
CatalysApplication application = runtime.getCatalysApplication();
Properties properties = application.getProperties();
for (Iterator iterator = properties.keySet().iterator(); iterator.hasNext();)
{
    String key = (String)iterator.next();
    System.out.println(key + "=" + properties.getProperty(key));
}
```

## 8.1.9. JMX Component Operations

If you require more sophisticated JMX instrumentation (i.e. custom attributes read and written directly with their own customized descriptions, and custom operations available to the JMX client), then it is possible to supply custom JMX configuration. In this case, each component attribute appears as its own individual field on the Dashboard.

To do this, a number of extra configuration classes must be created in the *configuration* sub-package of the original component. Please follow the examples provided in the following directories of the Node installation.

- *templates/src/com/camerontec/catalys/server/test/*
- *templates/src/com/camerontec/catalys/server/test/configuration/*

Refer to the *JMXCustomComponent\*.java* files.

In addition to creating the configuration classes, the component must have a DTD definition as explained in the next section.

## Custom XML Elements

Typically custom components are configured by using a generic configuration element and providing the complete class name as an argument, e.g. *GenericProcessor*. As an alternative, you can create well-defined configuration elements that correspond to a specific custom class. This is necessary in order to support advanced JMX configuration described in this section.

Each Node configuration file must reference a Document Type Definition (DTD) file, which by default is *CatalysConfig.dtd*:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Config SYSTEM "CatalysConfig.dtd">
<Config>
    ...
```

```
</Config>
```

This DTD file is located in the *dtDs* directory of the installation. Also in that directory is *CustomElements.dtd*. To define your own custom elements, add them to this file.

```
<?xml version="1.0" encoding="UTF-8" ?>

<!-- Include master dtd file -->
<!ENTITY % CatalysConfig SYSTEM "CatalysConfig.dtd">
%CatalysConfig;

<!ELEMENT MyProcessor ANY>
<!-- ATTLIST MyProcessor
      class CDATA #FIXED "com.example.MyProcessor"
      id CDATA #REQUIRED
      msgTypes CDATA #IMPLIED
      customProp CDATA #IMPLIED
-->
```

Now update your configuration files to use the custom DTD. You can then reference your custom component without a generic configuration element:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Config SYSTEM "/path/to/dtDs/CustomElements.dtd">
<Config>
  <Application id="app">
    <Sessions>
      <Session counterpartycompid="CLIENT" compid="BROKER" fixversion="4.4" heartbeat="60">
        <Connections>
          <SocketConnection id="sc" listenport="2003"/>
        </Connections>
        <SessionManager>
          <SourceMessageProcessors/>
          <ListenerMessageProcessors>
            <MyProcessor id="myProcessor" customProp="someValue"/>
          </ListenerMessageProcessors>
        </SessionManager>
      </Session>
    </Sessions>
  </Application>
</Config>
```

## Runtime Instrumentation

To add JMX runtime attributes and operations to a custom component, define a runtime MBean interface and implement this in the component. Then have the component register itself with the platform MBean server when it is opened.

For example, a message processor could be enhanced to count the number of messages that it has received. To instrument this for JMX, perform the following steps:

- Define a runtime MBean interface to access the attributes and operations:

```
public interface MyProcessorMBean
{
    int getMessageCount();
    void resetMessageCount();
}
```

- Implement the interface in the message processor
- Define a constructor for the message processor which takes a `ConfigurationAgent` parameter in order to retrieve the global runtime configuration
- Register the component with the platform MBean server when it is opened

```
import com.camerontec.catalys.core.message.IFIXMessage;
import com.camerontec.catalys.server.configuration.ConfigurationAgent;
import com.camerontec.catalys.server.processor.FIXListenerBase;
import com.camerontec.catalys.util.messaging.MessageEvent;

import javax.management.MBeanServer;
import javax.management.ObjectName;
import java.lang.management.ManagementFactory;

public class MyProcessor extends FIXListenerBase implements MyProcessorMBean
{
    private final ConfigurationAgent configurationAgent;
    private volatile int messageCount;

    public MyProcessor(ConfigurationAgent configurationAgent)
    {
        this.configurationAgent = configurationAgent;
    }

    @Override
    public boolean openComponent() throws Exception
    {
        final boolean wasOpen = super.openComponent();

        if (!wasOpen)
        {
            final String componentID = getAttribute("id").toString();
            final String appID = configurationAgent.getApplicationId();
            final MBeanServer mBeanServer = ManagementFactory.getPlatformMBeanServer();
            final String mbeanFormat =
                "com.camerontec:CatalysServer=\"%s\",name=\"%s\",type=\"CatalysServer.MessageProcessor\"";

            mBeanServer.registerMBean(
                this,
                new ObjectName(String.format(mbeanFormat, appID, componentID)));
        }

        return wasOpen;
    }
}
```

```

@Override
public void onMessageFromFix(MessageEvent event, IFIXMessage message)
{
    System.out.println(message);
    messageCount++;
}

public int getMessageCount()
{
    return messageCount;
}

public void resetMessageCount()
{
    messageCount = 0;
}
}

```

This will give the custom MBean a name which is consistent with the Catalys naming scheme.

## 8.2. CLI Commands

### 8.2.1. Introduction

The Catalys Node [command-line interface](#) (CLI) is extensible, allowing users to create their own commands in any [custom component](#). Common use cases for this are providing administrative control of custom components and printing information or statistics related to a custom component or other aspects of the Node (e.g. memory usage).

### 8.2.2. Implementation

The following steps should be taken when creating a custom command:

#### Implement the ICommandLineProcessor Interface

The custom component must implement the [ICommandLineProcessor](#) interface.

```
public class CustomComponent implements ICommandLineProcessor { ... }
```

#### Define Command IDs

CLI commands are identified by a unique integer value. These values must be provided when creating [Command](#) objects. Custom command IDs should start from 0xA to avoid conflicts with other built-in commands.

```
private static final int COMMAND_ONE = 0xA001;
```

```
private static final int COMMAND_TWO = 0xA002;
```

## Create the Command Objects

Create a [Command](#) object for each custom command. It is useful to store these in an array.

```
private static final Command[] COMMANDS = {
    new Command(COMMAND_ONE, "command_one", "comm_one",
        "<arg1>: call to command_one with one arg", "", true),
    new Command(COMMAND_TWO, "command_two", "comm_two",
        "<arg1> <arg2>: call to command_two with two args", "", true),
};
```

## Register the Command Objects

Register the `Command` objects with the [CommandLine](#) singleton.

```
static
{
    final ICommandLineProcessor clp = ComponentCommandLineProcessor.getInstance();
    for (final Command command : COMMANDS)
    {
        command.processor = clp;
    }
    CommandLine.getInstance().registerAndCheck(COMMANDS);
}
```

## Implement the Process Method

The `ICommandLineProcessor` interface has one `process(Command command, String[] args)` method that must be implemented. This is where you validate the command and its arguments, perform an action in your custom component and return user output in a [TableSet](#) object.

```
@Override
public TableSet process(final Command command, final String[] args) {
    final TableSet results = new TableSet();
    switch (command.id) {
        case COMMAND_ONE:
            results.header("Command One", "Message");
            if (args.length != 1) {
                result.add(command.name, false, "command_one requires one argument <arg1>");
            } else {
                // ***Perform command_one actions in CustomComponent***
                results.add(command.name, true, "command_one successful");
            }
            break;
        case COMMAND_TWO:
            results.header("Command Two", "Message");
            if (args.length != 2) {
```

```

        result.add(command.name, false, "command_two requires two arguments <arg1>, <arg2>");
    } else {
        // ***Perform command_two actions in CustomComponent***
        results.add(command.name, true, "command_two successful");
    }
    break;
default:
    results.add(command.name, false, "Unknown command");
    break;
}
return results;
}

```

## 8.3. Tasks

### 8.3.1. Introduction

The Catalys Node Scheduler feature supports a number of built-in tasks that can be scheduled at an application or session level. The Scheduler is extensible, which allows users to create their own tasks and schedule them like the built-in tasks.

### 8.3.2. TaskBase Implementation

Custom tasks that run at the application level should extend the [TaskBase](#) class. The [ConfiguredObjectRegistry](#) object will be injected into the constructor of this type of task. The following are the steps to take to create a custom `TaskBase` task.

#### Extend the TaskBase Class

```
public class CustomTask extends TaskBase { ... }
```

#### Define the Constructor

In the constructor for your class you can optionally inject the `ConfiguredObjectRegistry`. This gives you a reference to any configured component in the Node.

```

private final ConfiguredObjectRegistry configuredObjectRegistry;
public CustomTask(ConfiguredObjectRegistry configuredObjectRegistry) {
    this.configuredObjectRegistry = configuredObjectRegistry;
}

```

#### Implement the doTask() Method

The `TaskBase` parent class has one `void doTask()` method that must be implemented. This is where actions associated with the task are performed.

```
@Override
public void doTask() throws Exception {
    logger.info("Performing Custom Task");
    // Call task's actions here
}
```

## Configuring the Custom Task

Configuring the custom task is done the same as with built-in tasks. Set the full class with namespace when defining the custom task in the XML. All other configuration is the same.

```
<Tasks>
  <Task id="custom_task" class="com.camerontec.catalys.server.task.CustomTask"/>
</Tasks>
```

### 8.3.3. PartyTaskBase Implementation

Custom tasks that run at the session level should extend the [PartyTaskBase](#) class. The [ConfiguredObjectRegistry](#) and [IConnectionPointConfiguration](#) objects will be injected into the constructor of this type of task. The following are the steps to take to create a custom [PartyTaskBase](#) task.

#### Extend the PartyTaskBase Class

```
public class CustomPartyTask extends PartyTaskBase { ... }
```

#### Define the Constructor

In the constructor for your class you can optionally inject the [ConfiguredObjectRegistry](#) and [IConnectionPointConfiguration](#) objects. This gives you a reference to any configured component in the Node and passes a reference to the session the custom task is configured to run on.

```
public CustomPartyTask(ConfigurationObjectRegistry configurationObjectFactory,
    IConnectionPointConfiguration connectionPointConfiguration) {
    super(configurationObjectFactory, connectionPointConfiguration);
}
```

#### Implement the doTask() Method

The [PartyTaskBase](#) parent class has one `void doTask()` method that must be implemented. This is where actions associated with the task are performed.

```
@Override
```



```
public void doTask() throws Exception {
    ISessionManager session_manager = getSessionManager();
    // perform action on session
    logger.info(session_manager.getConnectionPoint().toString());
}
```

## Configuring the Custom Task

Configuring the custom task is done the same as with built-in tasks. Set the full class with namespace when defining the custom task in the XML. All other configuration is the same.

```
<Tasks>
  <Task id="custom_party_task" class="com.camerontec.catalys.server.task.CustomPartyTask"/>
</Tasks>
```

## 8.4. Persistent Objects

### 8.4.1. Introduction

This section is designed to give developers an understanding on how to use Catalys Node persistent objects. The Node provides built-in support for [FIX session persistence](#), so that sessions can recover from application re-starts. However, Node persistence is not just limited to session persistence. The Node provides mechanisms to allow persistence of any piece of data. This is achieved via the Node Collections framework.

Collections are groups of objects which are persisted to disk. Depending on how these objects need to be retrieved, they can be persisted as an indexed list collection or as a map collection. In addition there is a "high capacity" version of the list collection which gives better performance for high volumes of data.

Each persistent collection is associated with one directory on disk. In that directory are the persistence file(s) for that collection.

### 8.4.2. Collection Registry

Persistent collections are managed through the Collection Registry, which is a Catalys Node Service, identified by the name `CollectionRegistry`.

The Collection Registry service implements the [ICollectionRegistry](#) interface. This interface provides the methods necessary to access registered collections.

A custom implementation of this interface can be used as long as it is registered as a service with the name: `CustomCollectionRegistry`. If such a service is registered, the default service will delegate to this custom one, falling back to its own methods if the appropriate custom ones are not provided.

The collection registry can be accessed by the following:

```
ICollectionRegistry registry = (ICollectionRegistry)ServiceRegistry.getInstance()  
    .getService(CollectionRegistryService.SERVICE_NAME);
```

## Getting a Collection

Once the collection registry has been obtained, specific collections can be accessed. This is achieved via the Collection Registry `get*` methods, which retrieve a collection of the specified name, creating it if it does not already exist. There are variants for lists and maps. The caller supplies the classes that make up the elements in the collection, a single class in the case of a list, and a key class and a value class in the case of a map.

## List Collections

To get a list collection (a collection which is keyed by an integer index), call the [getList](#) method.

For example to get a list of type `Integer` use:

```
List<Integer> list = registry.getList("integerList", componentProperties, Integer.class);
```

There is also a variant of the [getList](#) method which takes a custom [externalizer](#).

## Map Collections

To get a map collection (a collection which is keyed by an object), call the [getMap](#) method.

For example to get a collection which maps `String` to `Integer`, use:

```
Map<String, Integer> map =  
    registry.getMap("someMap", componentProperties, String.class, Integer.class);
```

Like list collections, there is also a variant of the [getMap](#) method which takes custom [externalizers](#) for the key and value classes.

## Collection Properties

The properties provided when getting a list or map are used to configure the collection when it is created or opened. These properties are explained in detail in [Persistent Collections Properties](#). Most of the properties are common to all collections. The exceptions to this are:

- [isHighCapacityList](#): When true, `getList` will return a high-capacity list, which has specific [high-capacity properties](#). When false, `getList` will return a standard list which can be configured with [standard list properties](#). This property is not supported with maps.

- [isCachingEnabled](#) and [maxCacheElements](#) only apply to map collections.

## Name

The name used to identify and retrieve a collection must be unique across the collection registry. There are two standard [CollectionProperties](#) which can fulfill this requirement: the `name` attribute and the `directoryName` attribute. There is a [getName](#) convenience function provided which will choose `name` over `directoryName` if it is present in a properties object, and throw an exception if neither property is defined:

```
Map<K,V> map =
    registry.getMap(CollectionProperties.getName(properties), properties, K.class, V.class);
```

### 8.4.3. Compacting a Collection

Calling [compact](#) with the name of a collection will attempt to reduce the size of the collection. As the collections are persisted as journal files, compacting will remove all duplicate keys from the journal.

For example, there might be a task scheduled at the end of the day to do this.

```
registry.compact("routeMap");
```

### 8.4.4. Removing a Collection

Calling [remove](#) will remove a named collection from the registry. This operation will remove the underlying persistence files from disk. It is not replicated.

```
registry.remove("routeMap");
```

### 8.4.5. Collections

Once the collection has been obtained from the collection registry, it can be accessed and updated according to whether it is a list or a map.

It is important to note that changes to the collection are only persisted (and distributed in High Availability environments) when there is a direct action upon the collection. For example, a change to an element which has been retrieved from the collection will only be persisted once the element has been 'put back' into the collection. If the collection is a list, this means re-adding it; if it is a map it means re-putting it.

For example if a route table is being updated when an execution report is received for an order:

```
Route route = routingMap.get(clOrdId);
```

```
route.setConfirmedClOrdId(true);
//Without this line, the update to the map will not be persisted
routingMap.put(clOrdId, route);
```

It is possible to nest collections, but it also must be noted that the same restriction applies to collection updates as above. That is the outer collection must be updated for the modification to be persisted (and distributed).

## List Collections

For standard capacity lists, the `java.util.List` interface is fully supported. Elements can be added, retrieved, updated, removed at any index in the list.

For example as part of its end of day processing the router processor may delete all of the orders that have been replaced or canceled. It could do this by maintaining a deletion list, items from which are updated as they are processed.

```
List<String> deleteList = registry.getList("deleteList", componentProperties, String.class);
for (int i=0; i < deleteList.size(); i++) {
    DeleteRecord deleteRecord = deleteList.get(i);
    routingMap.remove(deleteRecord.getClOrdId());
    deleteRecord.setDeleted(true);
    deleteList.set(i, deleteRecord);
}
```

## High Capacity List Collections

The `java.util.List` contract is modified for high capacity list collections such that elements can only be added or removed from the end of the list, and the entire list can be cleared.

This example shows usage of a high capacity list in a component which stores `IFIXMessage` objects:

```
properties.setProperty(CollectionsProperties.IS_HIGH_CAPACITY_LIST, "true");
List<IFIXMessage> messages = registry.getList(
    CollectionProperties.getName(properties),
    properties,
    messageExternalizer);
...
messages.add(message);
...
messages.clear()
```

## Map Collections

The `java.util.Map` interface is fully supported for map collections. Elements can be put into the map, retrieved and removed from the map, the map can be iterated and cleared.

Note that due to the underlying map implementation, performing a `put()` or `putAll()` on the collection will result in a write to the disk regardless of whether the collection data has changed or not. Therefore it is the responsibility of the map's client to decide when it is appropriate to `put()` or `putAll()`.

This example shows usage of a map in a component which maps a string (a client order ID for example) to routes:

```
Map<String, Route> routeMap = registry.getMap(
    CollectionProperties.getName(properties),
    properties,
    String.class,
    Route.class);
...
// Add a new route
routeMap.put("newClOrdID", newRoute);
...
// Modify an existing route
Route route = routeMap.get("anotherClOrdID");
route.changeRoutingInformation(...);
routeMap.put("anotherClOrdID", route);
...
// Delete a route
routeMap.delete("id");
```

## 8.4.6. Externalizers

Externalizers serialize and deserialize an object to and from a [ByteSegment](#) in order to prepare it to be written to and read from persistent storage. `ByteSegment` objects are efficient low-level byte storage for the data that needs to be serialized. They minimize object creation, re-using their own pre-allocated storage where possible, and extending this storage only when necessary.

### Built-in Externalizers

We provide the following built-in externalizers. If custom externalizers are not specified, objects will be persisted using the most appropriate built-in externalizer.



#### Note

The data stored in the collection can be of type `ByteSegment` itself, minimizing the translation required for externalization. When this is the case there is no guarantee that object returned from persisted collections will be safe to store. i.e. object already returned may be changed on subsequent retrieval from a persistent collection.

Type	Externalizer
BytesSegment	<a href="#">BytesSegmentToBytesSegmentExternalizer</a>
String	<a href="#">StringUTFToBytesSegmentExternalizer</a>
Boolean	<a href="#">BooleanToBytesSegmentExternalizer</a>
Short	<a href="#">ShortToBytesSegmentExternalizer</a>
Integer	<a href="#">IntegerToBytesSegmentExternalizer</a>
Float	<a href="#">FloatToBytesSegmentExternalizer</a>
Long	<a href="#">LongToBytesSegmentExternalizer</a>
Double	<a href="#">DoubleToBytesSegmentExternalizer</a>
Object	<a href="#">ObjectToBytesSegmentExternalizer</a> (see <a href="#">below</a> )

## ObjectToBytesSegmentExternalizer

Any object which implements `java.io.Serializable` can be handled by this externalizer. This default object serialization uses `ObjectOutputStream` and `ObjectInputStream`. If the class also implements `java.io.Externalizable` then the serialization is controlled by the `writeExternal` and `readExternal` methods of this interface. If this is still insufficient for an object's serialization, then a [Custom Externalizer](#) can be written.

## Custom Externalizers

A custom externalizer may be required if object-based serialization is not efficient enough for the application. A custom externalizer must implement the [IBytesSegmentExternalizer](#) interface. This interface has two methods:

Method	Notes
<a href="#">toBytesSegment</a>	Serializes the object to its <code>BytesSegment</code> representation.
<a href="#">fromBytesSegment</a>	Creates an object by deserializing the specified <code>BytesSegment</code> .

There is a base implementation, [BytesSegmentExternalizerBase](#), which performs handling of null values for `BytesSegment` and object parameters.

There is a further implementation, [ConcurrentStreamingBytesSegmentExternalizerBase](#), which provides support for concurrency as well as wrapping the underlying `BytesSegments` in

[BytesSegmentDataInputStream](#) and [BytesSegmentDataOutputStream](#), taking care of their creation and wrapping.

## 8.5. Message Factories

### 8.5.1. Introduction

The implementation steps below show how to develop your own message factory class. Refer to the [Message Factories](#) section for information on typical use-cases, a list of provided message factories and configuration examples.

### 8.5.2. Implementation

To create and configure a custom message factory:

#### Implement the IFIXMessageFactory Interface

The custom message factory must implement the [IFIXMessageFactory](#) and [IConfigurable](#) interface.

```
public class CustomMessageFactory implements IFIXMessageFactory, IConfigurable { ... }
```

#### Implement the createMessage(...) Methods

The [IFIXMessageFactory](#) interface has two `createMessage(...)` methods that must be implemented. This allows you to create messages of a certain type as well as add non-standard fields to certain message types.

```
@Override
public IFIXMessage createFIXMessage() {
    IFIXMessage message = new FIXMessageAsArray();
    if (dictionaryName != null) {
        message.setDictionaryName(dictionaryName);
        message.setUseExactFixDictionary(useExactFixDictionary);
    }
    return message;
}

@Override
public IFIXMessage createFIXMessage(String msgType) {
    IFIXMessage message = createFIXMessage();
    message.setValue(Field.MsgType.tag, msgType);
    // add field1 and field2 to Logout messages
    if (MsgType.Logout.value.equals(msgType)) {
        message.setValue(9001, this.field1Value);
        message.setValue(9002, this.field2Value);
    }
}
```

```
return message;
}
```

## Implement the configure(...) Method

The `IConfigurable` interface has `configure(...)` method that must be implemented. This allows you to access custom properties defined in the message factory configuration.

```
@Override
public void configure(IConfiguration configuration) throws ConfigurationException {
    Properties props = configuration.getConfigurationAsProperties();
    this.field1Value=props.getProperty("prop1");
    this.field2Value=props.getProperty("prop2");
    try {
        initialize(props);
    } catch (Throwable e) {
        throw new ConfigurationException("Message factory " + configuration.getRegistrationName()
            + " cannot be configured", e);
    }
    // Save this configuration as the last valid configuration used
    this.configuration = configuration;
}
```

## Implement Other Methods

The `IFIXMessageFactory` interface has more dictionary related methods that must be implemented. Refer to the [Javadoc](#) for details.

# 8.6. Message Transformers

## 8.6.1. Introduction

The implementation steps below show how to develop your own message transformer class. Message transformers are used by technology adapters to transform inbound FIX message objects into the data object that the adapter requires and vice versa. All of Catalys Node technology adapters have a default transformers but there are situations when a custom transformer is required.

## 8.6.2. Implementation

To create a custom message transformer:

### Implement the IMessageTransformer Interface

The custom message transformer must implement the [IMessageTransformer](#) interface.

```
public class CustomMessageTransformer implements IMessageTransformer { ... }
```



## Message Transformer Methods

The `setAttributes(...)` exist to pass attributes to the custom transformer.

The `toStream(...)`, `toObject(...)` and `toObjectList(...)` methods exist to convert `IFIXMessage` objects into the type of Java Object that the adapter requires.

The `toMessage(...)` methods exist to convert message objects from the adapter into `IFIXMessage` message objects.

The transformer should implement the methods it is required to support and throw unsupported exceptions for implementations it does not support as follows:

```
public List<Object> toObjectList(IFIXMessage[] messages)
    throws UnsupportedOperationException {
    throw new UnsupportedOperationException("Method toObjectList() not implemented.");
}
```

## Implement the IMessageProcessorEventTransformer Interface

To transform event messages for FIX connection status changes, session resets and exception events, you may optionally implement the [IMessageProcessorEventTransformer](#) interface.

```
public class CustomMessageTransformer implements
    IMessageTransformer, IMessageProcessorEventTransformer { ... }
```

### 8.6.3. Configuration

To configure a custom message processor on a technology adapter, set the `transformer` attribute on adapter element you are using to the full namespace and class name of your transformer. Every technology adapter is required to support this attribute.

## 8.7. Repeating Groups



### Important

As of Catalys 2.0, the legacy repeating group API that used the `RepeatingGroup` and `RepeatingGroupAccessor` classes is deprecated, and may be removed altogether in a later release. Client code uses those classes should migrate to the new methods demonstrated below.

## 8.7.1. Introduction

The `IFIXMessage` message object interface can contain repeating groups. The following examples demonstrate how to add a repeating group to a message object as well as how to parse an existing repeating group on a message object.

## 8.7.2. Adding a Repeating Group

To add a repeating group to a message object simply call the appropriate `addField()` method in the order that the repeating groups are required to be in, starting with the "NoOfID" FIX field. The order of the fields will be maintained within the message.

The following is an example of adding a repeating group with two entries. The first entry has a nested repeating group with two additional entries:

```
...
// Instantiate IFIXMessage object for a NewOrderSingle
final IFIXMessage msg = getToFIXMessageFactory().createFIXMessage(MsgType.NewOrderSingle.value);

// Start with Num in group tag.
msg.addField(Field.NoPartyIDs.tag, 2);

// Group Entry 1
msg.addField(Field.PartyID.tag, "ABC");
msg.addField(Field.PartyIDSource.tag, "SOURCEA");
msg.addField(Field.PartyRole.tag, "ROLEX");

// Nested Group
msg.addField(Field.NoPartySubIDs.tag, 2);
msg.addField(Field.PartySubID.tag, "BIC12345");
msg.addField(Field.PartySubIDType.tag, "NESTED1");
msg.addField(Field.PartySubID.tag, "BIC6789");
msg.addField(Field.PartySubIDType.tag, "NESTED2");

// Group Entry 2
msg.addField(Field.PartyID.tag, "DEF");
msg.addField(Field.PartyIDSource.tag, "SOURCEB");
msg.addField(Field.PartyRole.tag, "ROLEZ");
...
```

## 8.7.3. Parsing Repeating Groups

To parse a repeating group on a message object, iterate the message starting at the "NoOfID" FIX field of the repeating group. From there, call the `next()` method until `hasNext()` returns false. The following example is for parsing a `NoPartyIDs` repeating group.

```
// Instantiate member variables for parsing
private final ITagValue partyRepeatingGroupReturnBuffer = new StandardTagValue();
private ListIterator<ITagValue> partyRepeatingGroupIterator;
```

```

...
// Used to initialize the iterator
private ListIterator<ITagValue> updateIterator(IFIXMessage message,
                                             ListIterator<ITagValue> iterator,
                                             int tag,
                                             ITagValue returnBuffer) {

    // We first check to see whether the iterator may be bound to this message.
    if (message.isBindable(iterator)) {
        // Bind our reusable iterator to the message. Iteration will commence from the
        // specified tag (if it doesn't exist exception is thrown) or the first field
        // if the specified tag is <= 0.
        return message.bind(iterator, tag, returnBuffer);
    } else {
        // This iterator cannot be bound to this message. We have no choice but to create
        // a new iterator which is guaranteed to be bindable. We can save that one for
        // the next message and try to reuse it then.
        return message.iterator();
    }
}

//Called when inbound messages are received
@Override
public void onMessageFromFix(MessageEvent event, IFIXMessage message) throws Exception {
    // Initialize the reusable iterator and return value buffer
    partyRepeatingGroupIterator = updateIterator(message,
                                                partyRepeatingGroupIterator,
                                                Field.NoPartyIDs.tag,
                                                partyRepeatingGroupReturnBuffer);

    // Iterate over all fields using in a reusable iterator from a specific tag.
    if (message.hasField(Field.NoPartyIDs.tag)) {
        logger.info("Reusable iteration from Party Group:");
        while (partyRepeatingGroupIterator.hasNext()) {
            ITagValue field = partyRepeatingGroupIterator.next();
            logger.info("-> {}: {}", Field.from(field.getTag()), field.getValue().getValueAsString());
        }
    }
}

```

## 8.8. JMX API Advanced Operations

### 8.8.1. Configuring via JMX

All Catalys configuration elements have been instrumented for JMX. In addition, to support the ease of configuration via JMX, advanced operations have been added. These advanced operations fall into the following categories:

- **Addition Operations:** There are operations for the addition of each configuration element. These `addXXX` operations are executed on the particular MBean for which the XXX element is a child. Once the element has been added to the configuration, a new MBean is created, which can be edited and operated upon. There are two variants of some addition operations, the `addXXX` variant

where the component does not have any custom JMX configuration, and the `addCustomXXX` variant where the component does have custom JMX configuration. For the second variant, the additional `configurationPackageName` parameter must be supplied. This is the location of the Java package where the configuration class resides (this must be in the same package as the runtime class, or in a configuration sub-package). For further information on JMX configuration customization, refer to the [Programming Guide](#).

The `class` attribute of elements added using the non-custom variant must be set before the element can be reloaded into runtime, and for some MBeans additional configuration is also required. Then the configuration can be loaded into runtime by one of the `reload` operations (either `Application` or `Session`).

For those elements which can have custom properties, there is an operation called `addChildProperties`, which takes `id` and `refid` parameters, where the `refid` parameter is used to refer to another already defined properties element.

- **Copy Operations:** These are convenience operations that group together all of the addition operations that are required to duplicate an existing element.
- **Move Operations:** These are operations which move elements from one place in the configuration hierarchy to another.
- **Deletion Operations:** Most MBeans have a `markForDelete` operation. To delete the element, execute this operation, then reload the session or the application. The element will be deleted from the configuration and runtime. The description of this operation is not repeated below (except in the cases where the MBean does not have a `markForDelete` operation), as its functionality is the same for each different MBean.
- **Maintenance Operations:** These operations allow the configuration to be saved, reloaded, undone and diagnosed. They do not create or delete any MBeans. All MBeans have a `showConfigurationAsProperties` operation which returns a formatted string with a list of all of the MBean's attributes and their values.

Note that the configuration elements belonging to a `HighAvailabilityCluster` are not instrumented to be changed at runtime. These elements are read-only. Consequently there are no advanced operations for the High Availability components of a Catalys configuration.

## Available Configuration MBeans

The following configuration MBeans are available in the Node. For details about the operations of each MBean, follow the link to each Javadoc.

- [Configuration.Application](#)
- [Configuration.Application.Link](#)
- [Configuration.Application.MessageProcessingTemplate](#)
- [Configuration.Application.MessageProcessor](#)

- [Configuration.Application.Properties](#)
- [Configuration.Application.RulesPack](#)
- [Configuration.Application.SchedulingDefinitions](#)
- [Configuration.Application.SchedulingDefinitions.EventList](#)
- [Configuration.Application.SchedulingDefinitions.Schedule](#)
- [Configuration.Application.Selector](#)
- [Configuration.Application.Session](#)
- [Configuration.Application.Session.SessionManager](#)

## 8.8.2. Monitoring via JMX

Any Java application can interact with the [Management Agent](#) (MA) through the standard JMX API for the purpose of monitoring and managing the applications it exposes. This is in fact what JConsole does.

The following example program can serve as a blueprint for a custom monitoring program. It illustrates how to access the MA's MBeans, which are registered in the `com.camerontec` domain. Before running, the MIS, LMA, and Node must be started.

```
import javax.management.MBeanServerConnection;
import javax.management.ObjectName;
import javax.management.remote.JMXConnector;
import javax.management.remote.JMXConnectorFactory;
import javax.management.remote.JMXServiceURL;

public class MAAccessExample
{
    /**
     * Establishes a connection with the MA and displays the MBeans that are
     * available for monitoring.
     */
    public static void main (String[] args) throws Exception
    {
        String serviceURL = "service:jmx:jmxmp://localhost:10000";
        JMXConnector maConnector = JMXConnectorFactory.connect(new JMXServiceURL(serviceURL));
        maConnector.connect();
        MBeanServerConnection mbeanServerConnection = maConnector.getMBeanServerConnection();

        for (Object o: mbeanServerConnection.queryNames(new ObjectName("com.camerontec:*"), null))
        {
            ObjectName objectName = (ObjectName)o;
            System.out.printf("%s\n", objectName);
        }

        maConnector.close();
    }
}
```

For more information on the types of MBeans available, refer to [JMX MBean Reference](#)

## 8.9. Authentication

### 8.9.1. Introduction

The Node Authentication component enables a pluggable authentication module (PAM) style framework for the authentication of FIX counterparties via the FIX Logon message. The implementation is based on the standard [Java Authentication and Authorisation Service \(JAAS\)](#).

Apart from a simple template for demonstration purposes, this package does not ship with any login modules as it is expected that the requirements for these will be mostly customer specific. However, the source code for the demonstration login module is included for reference.

Note that Oracle supplies a number of standard alternatives for storing user information. These include:

- JndiLoginModule
- KeyStoreLoginModule
- Krb5LoginModule
- NTLoginModule
- UnixLoginModule

The Node ships with a simple implementation which takes user information from a specified XML file: [XmlLoginModule](#).

See the standard [Oracle documentation on Login configuration files](#) for more details.

### 8.9.2. References

Since the FIX authentication implementation is based on standard set of Java APIs, this document references Javadocs for various interfaces and classes. It is recommended that the Javadoc for referenced interfaces and classes be read in conjunction with this documentation.

The most important of these are listed below:

- [Configuration](#)
- [LoginContext](#)
- [LoginModule](#)
- [Callback](#) and its standard implementations used by this package
  - [NameCallback](#)
  - [PasswordCallback](#)

- [TextOutputCallback](#)

### 8.9.3. FIX Authentication Implementation

FIX counterparty authentication is handled by a [custom monitor](#) designed specifically for this purpose.

When configured the `FixLoginAuthenticationMonitor` does the following:

- Intercepts each FIX Logon message, creates a [LoginContext](#) and calls its associated [login](#) method.
- On authentication failure, the FIX Logon attempt is rejected and the socket connection with the counterparty is dropped.
- On authentication success the [LoginContext](#) is stored until the FIX session is lost, at which point a [logout](#) is initiated.

Exactly how a FIX Logon message is used in the authentication process depends on which [LoginModule](#)(s) are used. See the sections below on [login module configuration](#) and [support for custom login module implementations](#) for more details.

### 8.9.4. FIX Authentication Configuration

The `FixLoginAuthenticationMonitor` can be defined as a [CustomMonitor](#).

#### FixLoginAuthenticationMonitor Attributes

For a complete list of attributes, see the [FixLoginAuthenticationMonitor Configuration Reference](#).

### 8.9.5. Login Module Configuration

The templates shipped with this package make use of the default [Configuration](#) implementation supplied with the JDK. See the [documentation](#) for the required configuration file format.

A custom configuration provider may be plugged in which retrieves login module in a completely different manner by changing the value of the `login.configuration.provider` security property (in the Java security properties file `<JAVA_HOME>/lib/security/java.security`).

The executable templates supplied with the FIX authentication package contain login configuration files which conform to the file format supported by the default [Configuration](#) implementation.

### 8.9.6. Support for Custom Login Module Implementations

Custom login modules must implement the [LoginModule](#) interface. See the [documentation](#) for the implementation contract.

In order to facilitate custom login module implementations the following callbacks are supported:

- [NameCallback](#) - login modules use this callback to obtain the username from the FIX Logon message
- [PasswordCallback](#) - login modules use this callback to obtain the password from the FIX Logon message
- [TextOutputCallback](#) - login modules use this callback to communicate arbitrary text back to the authenticator
- [TagValueCallback](#) - login modules use this callback to obtain the value of any arbitrary tag in the FIX Logon message
- [ConnectionPointCallback](#) - login modules use this callback to obtain the [IConnectionPoint](#) associated with the FIX session on which the Logon message was received.

### 8.9.7. FIX Authentication Templates

There are two separate templates shipped with the FIX authentication package under *templates/Authentication*:

1. **Simple** - Uses a single login module.
2. **Stacked** - Uses multiple stacked login modules.

See the *ReadMe.txt* files in each of the respective template directories for more details.



# Chapter 9. Performance Tuning

## Table of Contents

9.1. Introduction .....	374
9.2. Measuring Performance .....	375
9.2.1. Measuring Throughput .....	375
9.2.2. Measuring Latency .....	376
9.2.3. Instrumenting Message Counts .....	377
9.3. Hardware Considerations .....	378
9.3.1. CPUs and Threading .....	378
9.3.2. Memory .....	378
9.3.3. Disk Storage .....	378
9.3.4. Network .....	378
9.4. Configuring for Performance .....	379
9.4.1. Threading .....	379
9.4.2. Logging .....	383
9.4.3. Message Validation .....	383
9.4.4. Persistence .....	383
9.4.5. Choosing an IFIXMessage Implementation .....	383
9.5. JVM Tuning .....	384
9.5.1. JVM Configuration .....	384
9.6. Custom Code .....	385
9.6.1. Message Object Reuse .....	385
9.6.2. Efficient Field Retrieval .....	386
9.6.3. Floating-point Types .....	387

## 9.1. Introduction

The performance of the Catalys Node, or any FIX engine, is typically expressed by two key metrics: *throughput* and *latency*. Throughput indicates how many messages pass from one point to another in a given time interval, and latency indicates the time it takes for a single message to pass between two points.

How well the Node performs in these two aspects depends on several factors, including hardware, configuration, message load and downstream application processing. The Node provides several features and parameters that can be tuned specifically for your environment and message processing characteristics. However, before making any configuration adjustments, it is a good idea to benchmark the Node's performance in your environment and apply optimizations incrementally to gauge the relative benefit.

## 9.2. Measuring Performance

The standard Node installation includes tools that allow you to measure the throughput and latency of messages passing through the engine. You may also find it useful to measure performance using your own tools, either within the application or externally.

For working demonstrations of the examples illustrated below, please see the templates in the *templates/Benchmark* directory.

### 9.2.1. Measuring Throughput

The built-in `PerformanceMessageTester` is a message processor that measures throughput in terms of messages per second. It can measure inbound or outbound throughput and can be nested at any level in the message processing chain. This flexibility allows you to measure throughput at specific points in the engine, which is useful for isolating sources of poor performance.

To configure inbound throughput measurement:

```
<Session counterpartycompid="SEND" compid="RECV" fixversion="4.2">
  <Connections>
    <SocketConnection id="sc" listenport="5001"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors/>
    <ListenerMessageProcessors>
      <PerformanceMessageTester id="pmt" cycles="10000"/>
    </ListenerMessageProcessors>
  </SessionManager>
</Session>
```

To configure outbound throughput measurement:

```
<Session counterpartycompid="RECV" compid="SEND" fixversion="4.2">
  <Connections>
    <SocketConnection id="sc" hostname="localhost" port="5001"/>
  </Connections>
  <SessionManager>
    <SourceMessageProcessors>
      <PerformanceMessageTester id="pmt" cycles="10000">
        <!-- message generator -->
        <SampleMessageSource id="sms" interval="1" checkBeforeSend="true"/>
      </PerformanceMessageTester>
    </SourceMessageProcessors>
    <ListenerMessageProcessors/>
  </SessionManager>
</Session>
```

The `PerformanceMessageTester` outputs its measurements to the Node's log file:

```
INFO (Inbound) Last: 100000 messages, 4437ms, 22537.75 messages/sec
INFO (Inbound) Total: 100000 messages, 4437ms, 22537.75 messages/sec
INFO (Inbound) Last: 100000 messages, 3240ms, 30864.2 messages/sec
INFO (Inbound) Total: 200000 messages, 7677ms, 26051.84 messages/sec
INFO (Inbound) Last: 100000 messages, 3168ms, 31565.66 messages/sec
INFO (Inbound) Total: 300000 messages, 10845ms, 27662.52 messages/sec
```

## 9.2.2. Measuring Latency

The Node's native FIX message representation, `IFIXMessage`, contains a timestamp property that stores time expressed in nanoseconds. Latency within the Node is measured by setting this timestamp at a certain point in the message path and then comparing it with the current time at some later point.

Using the built-in `Timestamp` and `Timerecord` configuration elements, you can control where the starting and ending timestamps are taken, whether on inbound or outbound messages.

The `Timerecord` element will collect the measurements using a class that implements `IAccumulator`. We provide the following implementations:

- [HistogramAccumulator](#) stores the samples and groups them into equally sized bins to form a histogram, which provides a distribution of message latencies.
- [AggregatingStatisticalAccumulator](#) aggregates the samples and keeps track of the average and standard deviation.
- [StoringStatisticalAccumulator](#) stores the samples and apart from average and standard deviation, it can provide information about median and percentiles.

To measure inbound message latency using the `HistogramAccumulator`:

```
<Session counterpartycompid="SEND" compid="RECV" fixversion="4.4">
  <Connections>
    <SocketConnection id="sc" listenport="8000"/>
  </Connections>
  <SessionManager>
    <!-- Timestamp incoming FIX message -->
    <Timestamp id="tsInbound" />
    <SourceMessageProcessors/>
    <ListenerMessageProcessors>
      <NullMessageProcessor id="placeholder">
        <Timerecord id="trInbound"
          class="com.camerontec.catalys.util.stats.HistogramAccumulator">
          <Properties id="props">
            <Property name="dumpEvery" value="10000"/>
            <Property name="name" value="inbound" />
          </Properties>
        </Timerecord>
      </NullMessageProcessor>
    </ListenerMessageProcessors>
  </SessionManager>
</Session>
```

To measure outbound message latency using the `HistogramAccumulator`:

```
<Session counterpartycompid="RECV" compid="SEND" fixversion="4.4">
  <Connections>
    <SocketConnection id="sc" hostname="localhost" port="8000"/>
  </Connections>
  <SessionManager>
    <Timerecord id="trOutbound"
      class="com.camerontec.catalys.util.stats.HistogramAccumulator">
      <Properties id="props">
        <Property name="dumpEvery" value="10000"/>
        <Property name="name" value="outbound" />
      </Properties>
    </Timerecord>
    <SourceMessageProcessors>
      <!-- message generator -->
      <SampleMessageSource id="sms" interval="1" checkBeforeSend="true">
        <!-- Timestamp outbound FIX message -->
        <Timestamp id="tsOutbound"/>
      </SampleMessageSource>
    </SourceMessageProcessors>
    <ListenerMessageProcessors/>
  </SessionManager>
</Session>
```

The results are output to the Node's log file.

Each configured `Timerecord` is exposed as an MBean via JMX. They are located underneath `Catalys-Server.Session.TimeRecord` and are named corresponding to the ID assigned in the configuration file.

### 9.2.3. Instrumenting Message Counts

Message count statistics can be collected using the `MessageCountsStatsService`, which writes the statistics to a CSV file. For example:

```
TIMESTAMP,TOTAL.in,TOTAL.out,TOTAL.total,Party1~Party2.in,Party1~Party2.out,Party1~Party2.total
20110429141549,5,1,6,5,1,6
20110429141649,9,2,11,9,2,11
20110429141749,13,3,16,13,3,16
20110429141849,17,4,21,17,4,21
20110429141908,17,4,21,17,4,21
```

The service is configured as follows:

```
<Application id="app">
  <Services>
    <GenericService
      id="MessageCountsStatsService"
      class="com.camerontec.catalys.server.sessionmanager.MessageCountsStatsService">
    <Properties id="props">
      <Property name="sessionIdentifierNeutral" value="true" />
    </Properties>
    </GenericService>
  </Services>
</Application>
```

```

    <Property name="counterIntervalInMinutes" value="15" />
    <Property name="fileName" value="stats.csv" />
  </Properties>
</GenericService>
</Services>
</Application>

```

If the `sessionIdentifierNeutral` attribute is set to true, the actual session names are replaced with anonymous names. The `counterIntervalInMinutes` should be set to a value less than the number of minutes in 24 hours.

## 9.3. Hardware Considerations

### 9.3.1. CPUs and Threading

The Catalys Node is a multi-threaded application — it creates a [number of threads](#) per FIX session and several other threads depending on the configured options. Because of this, it is generally useful to run the Node on a server that has a large number of cores. This allows the OS to schedule the various threads on separate cores, thus reducing contention.

Several [configuration options](#) can be used to modify the Node's threading model, which may yield performance improvements in your environment. It is also possible to [reserve specific cores](#) for use by the Node.

### 9.3.2. Memory

The amount of RAM and heap space required by the Node is dependent upon how many FIX sessions are configured and which features are enabled. For an installation of 5 or fewer FIX sessions with typical message processing features, a heap size of 512MB is generally adequate.

### 9.3.3. Disk Storage

The Node's [logging](#) and [persistence](#) features perform a significant amount of disk I/O. A slow storage subsystem can create a performance bottleneck, therefore it is recommended to use the fastest storage option possible.

The majority of the Node's disk I/O operations are writes. If possible, your filesystem and storage subsystem can be tuned to favor write performance over read performance.

Message persistence and log data is likely critical to your operations. We recommend using a redundant storage subsystem, i.e. RAID, in order to protect against hardware failures.

### 9.3.4. Network

The amount of network capacity required by the Node depends on the message volume you expect to send and receive, and on the capacity of your upstream network. If the aggregate message volume of

one Node instance is extremely high, FIX sessions can be separated across several physical NICs by setting the `SocketConnection`'s [bindAddress](#) attribute (for acceptors) or the [localhost](#) attribute (for initiators).

When using the Node's [High Availability](#) feature, we recommend having a separate physical network reserved for HA communications. This allows for segregation of FIX traffic and HA traffic so that neither contends with the other. For more information see the [ClusterNode](#) element and its [localhost](#) attribute.

## 9.4. Configuring for Performance

### 9.4.1. Threading

The Node's threading model can be tuned to cope with different scalability and performance requirements. By default, each session creates four threads used for FIX message processing:

- **MessageAssembler:** Reads incoming message bytes from the session's underlying connection and assembles those bytes into `IFIXMessage` objects.
- **Protocol:** Manages the low-level FIX protocol and adds incoming messages to the application queue.
- **SessionManager:** Retrieves incoming messages from the application queue and passes them through each message processor.
- **Heartbeat:** Sends out heartbeats at regular intervals.

The first three threads process messages concurrently and relay messages via queues. This approach helps insulate each session from the performance characteristics of the other sessions. It also separates the reading of FIX messages from the processing of FIX messages, meaning that these things are less likely to interfere with each other.

If the queue between the protocol and user thread exceeds 2000 unprocessed messages, built-in throttling will be enabled. The protocol thread will sleep 10ms before placing additional messages on the unprocessed message queue. This will be disabled once the queue has less than 2000 messages. When throttling is enabled, the highest rate that inbound messages can be processed is 100/sec. The maximum number of queued messages and throttling rate are not configurable.

Unfortunately there is overhead involved in passing messages between threads, as well as in context switches caused by making blocking system calls and by operating system thread preemption. The following features can minimize this overhead and can be used to achieve significant improvements in latency, throughput and jitter:

- Instead of using multiple threads per session to process incoming messages, a session's `MessageAssembler` thread can process each message entirely, eliminating the need for intermediate queuing. See [Unqueued Mode](#).
- Instead of creating a separate `MessageAssembler` thread for each session, a single thread that uses multiplexed, non-blocking I/O can read messages for all sessions. See [Multiplexing Mode](#).

- CPUs can be reserved for exclusive use by certain critical threads. This reduces overhead caused by thread preemption and CPU contention. See [Reserving CPUs](#).

These three features are designed to work together and can be safely combined.

#### 9.4.1.1. Unqueued Mode

Unqueued mode is an alternate threading model which can decrease the per-message latency in the Node. This can prove useful if the observed latency is due to FIX session-layer processing, i.e. before FIX messages reach the application layer (message processors). It may also prove valuable even when the performance of the Node is deemed satisfactory. To enable:

```
<Application unqueuedMode="true" ...>
```

When operating in unqueued mode, the tasks usually performed by the Protocol and SessionManager threads are consolidated into the MessageAssembler thread, eliminating the intermediate queuing of messages. Messages are read from the underlying connection and delivered to messages processors in a single thread. By avoiding thread context switches and queuing overhead, overall processing is generally faster. Note that in some circumstances the Node may need to temporarily switch back to queued mode if FIX messages arrive out of sequence.

#### 9.4.1.2. Multiplexing Mode

Multiplexing mode is an application-wide configuration option that controls the way the Node reads FIX messages from network sockets. To enable:

```
<Application multiplexingMode="select|spin" ...>
```

Instead of using blocking single-thread-per-socket I/O, the Node will use multiplexed I/O to read from all FIX session sockets in a single thread.

Multiplexing mode can operate in two ways: *select* or *spin*. In select mode the reading thread can block if there is nothing to read on any socket. In spin mode the reading thread will never block (while reading) and will continually check for new data to read from the sockets.



#### Caution

Enabling spin mode may result in constant 100% CPU utilization. Therefore we recommend reserving a CPU as explained in the following section.

### 9.4.1.3. Reserving CPUs

Some operating systems provide the ability to reserve one or more CPUs for exclusive use by a given application or thread. The Node can be configured to *pin* threads to reserved CPUs, which may improve message delivery performance by reducing overhead from thread preemption and CPU contention by other processes. This often results in lower, more consistent message latency.

Only certain threads in the Node are "pinnable", as not all threads would realize a performance benefit. The following threads can be pinned:

MessageAssembler	One thread is created per session, only when not using multiplexing mode.
FIX-Reader	Created when using <a href="#">multiplexing mode</a> as an alternative to MessageAssembler.
HA-Messaging-Reader-Data	Created only when High Availability is configured. Requires the <a href="#">minimizeContextSwitchLatency</a> attribute of HighAvailabilityCluster to be enabled. <a href="#">More details</a> .
HA-Messaging-Writer-Data	

### Isolating CPUs on Linux

To isolate CPUs on Linux, the `isolcpus` kernel boot parameter must be configured. For example:

```
kernel /vmlinuz-2.6.18-194.el5 ro root=LABEL=/ isolcpus=2,4,6
```

Each integer corresponds to a CPU ID listed in `/proc/cpuinfo` that you would like to isolate. CPU 0 is usually reserved for the OS and should not be isolated.

IRQ balance must also be disabled:

```
chkconfig irqbalance off
```

The system must then be rebooted.

### Configuring Reserved CPUs in the Node

Once a number of CPUs have been identified and isolated at the OS level, the Node must be configured with a set of reservable CPUs. There are two ways to configure this: via a CPU pool, or by mapping CPUs to specific threads.



#### Important

Please note the following configuration restrictions:



- CPU 0 is usually reserved for the OS and should not be reserved. Configuring the Node to use it will result in a configuration error.
- When configuring a CPU pool and using [multiplexing mode](#), only one reserved CPU is needed for inbound message reading, regardless of the number of sessions.
- Mapping CPUs to specific threads is only supported when using multiplexing mode. Otherwise a pool of CPUs must be configured.

- **Reserving a pool of CPUs:**

When a pinnable thread starts, it will attempt to acquire one of the CPUs in the pool. If none are available, the thread will run unpinned.

```
<Application reservedCores="2,4,6" ...>
```

A range of CPU IDs can also be specified:

```
<Application reservedCores="2,4,6,8-10,13" ...>
```

- **Mapping CPUs to specific threads:**

With this option, a list of thread-to-CPU mappings can be configured. By putting a given thread on a specific CPU, you may realize performance benefits depending on the hardware configuration (e.g. putting related threads in the same NUMA zone).

```
<Application>
  <Threading>
    <Affinity>
      <ReservedCore thread="FIX-Reader" cpu="2"/>
      <ReservedCore thread="HA-Messaging-Reader-Data" cpu="4"/>
      <ReservedCore thread="HA-Messaging-Writer-Data" cpu="6"/>
    </Affinity>
  </Threading>
  <HighAvailabilityCluster id="cluster" minimizeContextSwitchLatency="true">
    ...
  </HighAvailabilityCluster>
  <Sessions>
    ...
  </Sessions>
</Application>
```

## 9.4.2. Logging

The Node can be configured to use our asynchronous, highly efficient binary logging subsystem. See the [Logging](#) documentation for more information.

In addition, some users may wish to disable the logging of sent and received FIX messages. See the [printMessages](#) and [printAdminMessagesOnly](#) configuration attributes.

## 9.4.3. Message Validation

The Node performs a number of validation operations on each FIX message received. In order to reduce processing times these can be disabled using [trustedMode](#). For example:

```
<Session trustedMode="true" ...>
```

## 9.4.4. Persistence

Persistence of FIX session state is an expensive operation. If disk-based persistence is not necessary in your environment, a memory-based persister can be used instead. For more information see [Session Persistence](#).

If disk-based persistence is necessary then the performance of the `JournalingPersister` can be improved by configuring the [useWriteBuffering](#) option. This option batches multiple writes to disk into a single operation, improving performance but reducing reliability. Recent transactions are not guaranteed to have been persisted if the engine goes down.

## 9.4.5. Choosing an IFIXMessage Implementation

[IFIXMessage](#) is the Node's internal representation of a FIX message. We offer several implementations of this interface, each with characteristics that are suited to different types of message processing. There is also good support for creating your own customized implementations.

The following table lists the available implementations and their characteristics:

Implementation	Characteristics	Recommended Usage
<a href="#">FIXMessageAs-IndexedByteArray</a>	Can be populated directly from a <code>byte</code> array, without the creation of new objects.	Inbound FIX messages and general purpose. The default implementation for inbound FIX messages.
<a href="#">FIXMessageAsArray</a>	Adding unordered fields is efficient, as are iterating,	Outbound FIX messages and inbound FAST messages.

Implementation	Characteristics	Recommended Usage
	clearing and retrieving fields by array index. Removing fields, adding fields in a specific order, and field lookup by tag require traversal of the array and are therefore less efficient.	The default implementation for outbound FIX messages.
<a href="#">FIXMessageAsList</a>	Implemented as an <code>ArrayList</code> and inherits all of the performance characteristics thereof. Retrieval of field by tag number requires list traversal.	When the use of <code>List</code> -like operations are necessary.
<a href="#">FIXMessageAsIndexedList</a>	Implemented as a <code>LinkedList</code> with an index by tag number to entries in the list. Retrieval of fields by tag is very efficient, but field removal requires a scan of the list to update the index.	When efficient tag retrieval is necessary.
<a href="#">OrderedFIXMessage</a>	Enforces field ordering on each message. The ordering is taken from the FAST templates.	Outbound FAST messages.

To configure a message factory other than the default, use a `ConfigurableMessageClassMessageFactory` element and set the `className` attribute to the class of the factory you would like to use. See the example in the [Message Factory](#) section.

## 9.5. JVM Tuning

### 9.5.1. JVM Configuration

The configuration of your Java Virtual Machine can influence the Node's performance. In particular, note the following parameters:

- Server mode (`-server`): Enables optimizations for long-running Java applications.
- Maximum heap space (`-Xmx`): Set to a value [adequate for the instance's needs](#).
- Initial heap space (`-Xms`): Set to the same value as `-Xmx` to avoid costly resizing of the heap at run time.
- Garbage collection algorithm: To avoid GC pauses where the engine appears unresponsive, choose a low latency ("concurrent") GC algorithm. A "parallel" as well as concurrent collector will take advantage

of multiple CPUs and perform multi-threaded GC. One such GC algorithm is the Concurrent Mark Sweep (CMS) collector for the Oracle JVM (`-XX:+UseConcMarkSweepGC`).

- Profiling and logging options (such as `-verbose:gc`, `-XX:+PrintGCDetails` and `-Xprof`): While useful in a test environment, these options slow down the VM so you should avoid using them in a production environment.

If deterministic GC behaviour and precise response times are of utmost importance in your environment, [Real Time Java](#) or 'pauseless' garbage collectors (such as AZUL Systems' Zing) may be useful.

## 9.6. Custom Code

The following optimizations may be useful when developing and tuning your custom Catalys components.

### 9.6.1. Message Object Reuse

Message reuse can improve performance when observing frequent garbage collections due to the creation of `IFIXMessage` objects. Ideally, the number of message objects created should be fixed regardless of the message volume. However, this is not always possible depending on how your component is using each message reference.

When reuse is not enabled, a new FIX message object is created for every inbound and outbound message. Unless a message object has been stored somewhere within the Node, it should be possible to reuse that same object for all subsequent messages. The question is how do we know if a message object has been stored?

Each Node message processor has a boolean attribute, `storesMessages`, that is `true` by default. In addition, each `IFIXMessage` has a `possiblyStored` attribute, which by default is `false`. Any message processor with `storesMessages` set to `true` will automatically set the `possiblyStored` attribute to `true` on all messages it processes.

After the Node has submitted a FIX message to all the relevant message processors, it checks the `possiblyStored` attribute. If `true`, it cannot reuse the message object, so a new one will be created. However, if `false` then that same message object can be used when processing the next FIX message.

The author of a message processor needs to decide whether or not the message processor stores message objects. If it does not, `storesMessages` should be set to `false` by calling the [setStoresMessages\(boolean\)](#) method from the message processor's constructor. This will enable reuse of FIX message objects when it's safe to do so, thereby reducing the amount of garbage creation.



## Note

Message reuse works only in [unqueued mode](#). In queued mode messages are stored and processed asynchronously by different threads, so message objects cannot be reused.

## 9.6.2. Efficient Field Retrieval

The [IFIXMessage](#) interface has type-specific accessors that take advantage of the message's native underlying data when possible, often avoiding unnecessary object creation. For example:

```
// wrong
String s = msg.getValueAsString(Constants.TAGiQuantity);
int quantity = Integer.parseInt(s);
```

Use the type-specific method instead, which avoids the `String` creation:

```
// correct
int quantity = msg.getValueAsInt(Constants.TAGiQuantity);
```

Also be mindful of unnecessary conversion between ASCII `byte` arrays and `String` objects. The raw ASCII data in inbound messages can be accessed directly without creating a `String` by using the [BytesSegment](#) class. This class provides a view to a portion of a `byte[]` and has sensible `hashCode` and `equals` methods which allow for comparison of raw ASCII data. Therefore it is more efficient to perform repetitive operations on ASCII values using the raw bytes instead of `String`. For example:

```
BytesSegment SYMBOL_CONSTANT = new BytesSegment("IBM".getBytes());
BytesSegment valueFromMessage = msg.getValueAsAsciiBytes(Constants.TAGiSymbol);
boolean eq = valueFromMessage.equals(SYMBOL_CONSTANT);
```

In situations where you don't need to know the data type, for example to simply copy the value from one place to another, access the data as an [IValue](#):

```
IValue value = source.getFieldValue(Constants.TAGiQuantity);
target.setValue(Constants.TAGiQuantity, value);
```

The `MsgType` and `MsgSeqNum` fields have their own accessor methods, `getMsgType()` and `getMessageSeq()`. These methods provide the most efficient way of retrieving their respective values.

### 9.6.3. Floating-point Types

Another very important type of data in FIX is the price. It might seem that the most natural way to represent a price in Java would be as `float` or `double`. However, these types introduce precision and rounding problems and inefficiencies when formatting as text.

The Node provides the [Decimal](#) class as an alternate representation of fractional numbers such as prices. `Decimal` objects represent a fractional number as two whole numbers, a mantissa and an exponent. For example, the price \$1.52 would be represented as a mantissa of 152 and an exponent of -2. Quantities can also be represented in this way.

`Decimal` has the following advantages:

- No precision/rounding issues
- Very efficient to serialize and format
- Directly compatible with the industry standard FAST

Methods are provided for performing basic arithmetic. These methods avoid the precision/rounding issues of `float` and `double` and are also very fast. For example:

```
Decimal price = msg.getValueAsDecimal(Constants.TAGiPrice);
Decimal qty = msg.getValueAsDecimal(Constants.TAGiQuantity);
Decimal consideration = price.multiply(qty);
```

# Appendix A. Configuration Reference

This appendix provides a complete reference of configuration attributes for all Catalys components.

The **mutability** of each attribute is one of the following:

- Fixed to default value: The value of the attribute cannot be changed.
- Restart required: The Node must be restarted after changing the value.
- Reactivation required: The component must be deactivated and reactivated after changing the value.

## AdvancedSocketAdapter

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.socket.advanced.- SocketAdapterSourceListener
msgTypes	
The FIX message types to process with this listener. If not set, all message types will be processed.	
Type:	String

Required:	No
Mutability:	Restart required
<b>port</b>	
Port on which the Advanced Socket Adapter listens for a client connection.	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	5005
<b>ackTimeout</b>	
The amount of time (in milliseconds) the server should wait for acknowledgment before it considers the connection dead and disconnects.	
If the Socket Adapter has sent a RECV_MESSAGE and has not received a SEND_ACK within ackTimeout then it will disconnect. If this value is set to 0 then the Socket Adapter does not wait for acknowledgment.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	10000
<b>overridePreviousClient</b>	
Controls whether a new client connection will cause the current one to be disconnected.	
If this is set to true, any client connecting with the ID of a currently connected client will cause the currently connected client to be disconnected. Otherwise, the newly connected client will be disconnected.	
Type:	String
Required:	No



Mutability:	Reactivation required
Default value:	false
Legal values:	true   false
<b>addDataToAckMsg</b>	
<p>Configures the contents of the Data field of the ack message.</p> <p>If set to 'none', the default value, the data field of the ACK message will be empty.</p> <p>If set to 'seqnum', the ACK message 'data' field will contain the outbound sequence number assigned to the message that is being acknowledged (if known). In the special case of SourceMessageBuffer message processors, the sequence number will not be set if the message was buffered because the session was not available to send the message directly.</p> <p>If set to 'fixmsg', the ACK message 'data' field will contain the message as it was sent out. This includes modifications made to the message by any intervening message processor and sequence number and datetime fields set by the session. If the message is not sent out and is buffered, the ACK message will contain the message as it was written to the buffer.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	none
Legal values:	none   seqnum   fixmsg
<b>broadcastAckToClients</b>	
<p>Controls whether acknowledgements will be broadcast to all connected socket clients.</p> <p>If set to false (the default), ACK messages are sent to the socket client application in response to outbound (out to the FIX session) protocol messages.</p> <p>If set to true, ACK messages will be sent to ALL socket client applications currently connected. In conjunction with addDataToAckMsg, this option allows a socket client application to eavesdrop on all outbound and inbound messaging.</p>	
Type:	String

Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

**directoryName**

The name of the directory used to persist incoming messages (from the FIX session).

This store is used for message recovery.

The advanced socket adapter persists its messages in a map of sequence number against message, in a directory given by the this attribute. This persisted map can be configured by supplying the advanced socket adapter with a nested Properties element containing Map Collection Properties (refer to [Persistent Collection Properties](#)).

The files in this directory are compacted as part of the application's end of day procedure, which can be [scheduled](#) or called via the [CLI](#)).

Type:	String
Required:	No
Mutability:	Restart required
Default value:	SocketAdapter_

**rejectIfNotConnected**

Controls the behaviour of the Advanced Socket Adapter when it has a message to send to the socket client but the client is not connected.

If set to true, a business reject will be sent back to the sender of the message.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false

Legal values:	true   false
<b>keepAlive</b>	
Controls whether the underlying socket is kept open even if messages are sent only sporadically.	
If true, SO_KEEPALIVE is set on the underlying socket which will cause the socket to remain open even if no messages are sent or received in a long period of time. This is useful for sessions that should remain up during the day with only a few messages sent or received per day.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false
<b>transformer</b>	
Class that converts inbound/outbound FIX messages and events to and from socket messages.	
Must implement <a href="#">IMessageTransformer</a> .	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.core.message.StandardFixTransformer
<b>messageStoreType</b>	
The type of store to use when persisting inbound FIX messages to disk.	
When set to <code>map</code> , the adapter uses a <code>Map</code> -based message store whose in-memory index grows with message volume.	
When set to <code>highCapacity</code> , the adapter uses a <code>List</code> -based message store that has a fixed memory footprint regardless of message volume. This message store type provides better performance for high-volume sessions.	

Note that persisted data is not preserved when switching from one store type to another.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	map
Legal values:	map   highCapacity

### autoForwarding

Controls whether messages are automatically forwarded to other listeners in the chain.

If true, messages are automatically forwarded.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

## Application

### id

Unique ID of the application.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	default

### encryptAlgorithm

Encryption algorithm	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	PBEWithMD5AndTripleDES
<b>class</b>	
Class of the application.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.CatalysApplication
<b>instanceDetectionPort</b>	
<p>Socket port to which Catalys Node binds in order to prevent duplicate instances from starting on the same host.</p> <p>Valid port values are between 1024 and 65535 inclusive. The socket is bound to this port on 127.0.0.1 only. If unspecified, the port is deterministically chosen from a range of 12000 to 12999 based on the Application ID and cluster configuration if applicable. If set to -1, instance detection will be disabled.</p>	
Type:	Integer
Required:	No
Mutability:	Restart required
<b>unqueuedMode</b>	
<p>Enables/disables message processing in a single thread for all sessions, without intermediate queuing.</p> <p>When enabled, latency is improved in some circumstances. See <a href="#">Configuring for Performance</a> for more detail.</p>	

## Configuration Reference

Type:	Boolean
Required:	No
Mutability:	Restart required
Default value:	false
<b>switchToNonQueuedTimeout</b>	
<p>The maximum amount of time (in milliseconds) to wait for input queue to be empty.</p> <p>The maximum amount of time the receiver thread will wait for input queue to become empty to be able to switch to non-queued mode. This effectively throttles new message processing until the messages from the input queue are processed. In scenario, when there is a lot of messages in the input queue (missed messages when connection was down) and at the same time high load of the new messages this improves the switch time from queued to unqueued mode and with that the latency.</p>	
Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	10
Minimum value:	0
<b>multiplexingMode</b>	
<p>Enables/disables inbound I/O multiplexing strategy for all sessions.</p> <p>When enabled, latency as well as CPU utilization is improved in some circumstances. See <a href="#">Configuring for Performance</a> for more detail.</p>	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	none
Legal values:	none   select   spin

**multiplexingModeSendTimeout**

Maximum amount of time to spend trying to send a single message in multiplexing mode.

0 to try an initial batch of writes, otherwise the number of milliseconds to keep trying if the initial batch of writes is not sufficient. See [Configuring for Performance](#) for more detail.

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	100
Minimum value:	0

**reservedCores**

Space- or comma-separated list of numeric CPU identifiers.

The list can be a combination of individual CPU IDs and ranges, e.g. 5,7,9-11.

FIX message reading threads will be assigned to the specified cores. See [Configuring for Performance](#) for more detail.

Type:	String
Required:	No
Mutability:	Restart required

**asynchronousWriteMode**

Enables/disables outbound asynchronous writes for all sessions.

When enabled, latency as well as CPU utilization is improved in some circumstances. See [Configuring for Performance](#) for more detail.

Type:	String
Required:	No
Mutability:	Restart required

Default value:	none
Legal values:	none   wait   spin
<b>asynchronousWriterBufferCapacity</b>	
Minimum capacity of global FIX session asynchronous write buffer.	
When multiplexingMode is enabled and this value is greater than 0 all FIX session writes are buffered and subsequently sent on a single, global, sending thread.	
Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	10000
Minimum value:	0
<b>asynchronousWriteRequestSize</b>	
Initial size of an asynchronous write request slot in the global FIX session asynchronous write buffer.	
All slots are pre-allocated at startup and subsequently re-sized on demand. A larger initial size reduces the likelihood of later re-sizing.	
Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	300
Minimum value:	0
<b>highCapacityIndexerNumberOfThreads</b>	
High Capacity Indexer number of threads. If greater than zero shared highCapacityIndexerNumberOfThreads threads will be used for store indexing. If zero there will be one indexer thread per store.	



Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	0
Minimum value:	0

### **highCapacityIndexerBufferCapacity**

High Capacity Indexer buffer capacity.

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	100000
Minimum value:	0

### **connectionPointMatcher**

Connection point matcher class.

Matches parties from FIX messages with those configured on a session. Using the defaults requires all fields of the connection point (CompID, LocationID, SubID) to match.

The alternative `com.camerontec.catalys.core.session.LooseConnectionPointMatcher` only requires CompID to match.

Note, that `com.camerontec.catalys.core.session.LooseConnectionPointMatcher` should not be used when there are multiple sessions configured with the same CompID.

Alternatively custom implementations of `com.camerontec.catalys.core.session.IConnectionPointMatcher` can be provided.

Type:	String
Required:	No
Mutability:	Reactivation required

Default value:	com.camerontec.catalys.core.session.DefaultConnectionPointMatcher
<b>customComponentScanningPath</b>	
Colon-separated list of packages in which to search for custom components.	
This list of packages is scanned at start-up for custom component classes, which are then exposed through the JMX configuration API. Such classes can then be used as parameters to JMX configuration API operations that create custom components.	
The default value corresponds to the Catalys server packages that contain public custom components.	
Special care must be taken when changing this attribute: avoid specifying packages that contain large numbers of classes, since this is likely to cause some significant overhead at start-up, because all classes in these packages are loaded and introspected. So, typically, packages like "com" or "org" should not be used.	
Specifying an empty string for this attribute turns off the classpath scanning at start-up, in which case the fully qualified class name is required when creating custom components at runtime through the JMX configuration API.	
Note, that if a custom component extends a custom parent class then the package of the parent class must also be included.	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	com.camerontec.catalys.server.messageselector:com.camerontec.- catalys.server.test:com.camerontec.catalys.server.marketdata.test:com.- camerontec.catalys.server.sessionmanager:com.camerontec.catalys.- server.rules.test
<b>printMessages</b>	
Controls whether input and output FIX messages are logged to the console as INFO messages.	
Type:	Boolean
Required:	No
Mutability:	Reactivation required

Default value:	true
<b>printAdminMessagesOnly</b>	
Controls whether only admin messages are logged to the console as INFO messages.	
Admin messages are login, logout, heartbeat, sequence reset, test request and resend request. If true, the "printMessages" attribute is ignored and only admin messages will be logged.	
Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false
<b>printFormattedMessages</b>	
Controls whether a message summary is logged in addition to the entire message.	
The format of the summary can be controlled by defining a MessageToStringFormat element with format type SUMMARY. This option is used only if printMessages is set to true. Note: Enabling this option may have negative impact on the performance of the engine.	
Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false
<b>console</b>	
Controls whether or not the command line is supported in the application's console.	
Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	true

**logDetailedInboundConnectionEvents**

Controls whether details about connections to the server will be logged.

If set to true, when a client connects to the Catalys Node Server on a listening port (say 2000), the relevant Connection Point details for the configured parties on that port are logged. The logs will look like:

```
ServerConnectionManager: Received connection from /127.0.0.1:54391 into local port 2000
ServerConnectionManager:2000 - These are the parties configured for listening local port 2000:
ServerConnectionManager:2000 - ApplicationID: sell
ServerConnectionManager:2000 - Local Party (compID, subID, locationID): Fats Fast Market, -, -
ServerConnectionManager:2000 - External Party (compID, subID, locationID): Bigbroker, -, -
```

These logs are useful for sysadmins to determine the other end of a connection should it not succeed to send the Logon message.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

**stuckSessionTimeout**

Defines how long (in milliseconds) a session can be blocked writing an outbound message to the socket before it is considered stuck.

When set to zero (the default), the session is never considered stuck.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	0
Minimum value:	0

**disconnectOnStuckSession**

Determines whether the application closes the connections of sessions that are stuck trying to write an outbound message to a socket.

If not set, connections are not dropped.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### **licenseWarningPeriod**

The period (in days) before the license is due to expire, in which warning messages will be printed.

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	20
Minimum value:	0

### **preloadSessionDictionaries**

Controls whether to preload FIX dictionaries at startup instead of when they are required at runtime.

Loading a dictionary consumes a large amount of memory and takes a significant period of time. If this attribute is true then the delay will happen at application startup. If this attribute is false then this delay will occur during the processing of the first received message.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	true

### **maxAnonymousInboundConnections**

Defines the maximum number of allowable connections which have not been resolved to a configured FIX session.

Connections become resolved (and cease to be anonymous) when a valid FIX logon is received. This logon must match a configured session. When the limit is reached, new inbound connections are dropped immediately. Checking is disabled unless the value of this attribute is positive.

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	-1

#### **anonymousInboundConnectionTimeout**

The time in seconds an inbound connection may remain connected but not resolved (anonymous).

Anonymous connections are dropped after this time. Checking is disabled unless the value of this attribute is positive.

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	60

#### **scheduledEvents**

Assigns a scheduled events configuration.

Type:	Reference
Required:	No
Mutability:	Reactivation required

## **AsxOuchFileReferenceDataProvider**

**id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

#### **class**

Java class of the Runtime object corresponding to this configuration object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.asx.ouch.dataproviders.- AsxOuchFileReferenceDataProvider

#### **fileName**

Path to the file with static ASX Reference Data

Type:	String
Required:	Yes
Mutability:	Reactivation required

## AutoRoutingResponse

#### **id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
-------	--------

Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.rules.actions.AutoRoutingResponse
<b>msgTypes</b>	
The FIX message types to process. If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required
<b>directoryName</b>	
The name of the directory in which FIX session traffic is persisted so that it is not lost when the Catalys Node restarts.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>orderStatusesToArchive</b>	
<p>A list of order status values that indicate that an order is closed and can be archived.</p> <p>Archived orders are stored on disk, so that in-memory storage of orders does not grow without bound. If the node is re-started, this archive is deleted, and re-created from persistence files (if persistence is configured). If no statuses are specified then the following default list is used: Canceled, Expired, Done</p>	



for Day, Filled, Rejected. Value NONE means no archiving. These values must match the allowable values for the OrdStatus FIX field (FIX tag 39).

Type:	String
Required:	No
Mutability:	Reactivation required

### endOfDayClearingCondition

A Rules Expression specifying a condition which determines which orders to purge from the routing storage at end of day.

This parameter allows specification of a condition or set of conditions that determine whether a message will be cleared at the end of the day. This specification can be any expression allowed by the [Rules Engine Expression Language](#).

The following value clears orders which were only valid for the day (that is they do not have a TimeInForce field, or its value is set to zero): "isFieldPresent(TimeInForce) || (isFieldPresent(TimeInForce) && TimeInForce == 0)"

This example clears orders that have closed (that is, the value of their OrdStatus field is one of: Canceled, Filled, Expired, Done for Day, or Rejected): "matches(OrdStatus, '[42C38])"

These expressions can also be combined logically.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	matches(OrdStatus, '[42C38])    !isFieldPresent(TimeInForce)    TimeInForce == 0

## BecomingPrimaryScript

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required
<b>command</b>	
The location of the script to run when becoming the primary.	
Type:	String
Required:	Yes
Mutability:	Restart required

## BistltchGlimpseReferenceDataProvider

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of the Runtime object corresponding to this configuration object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.bist.ouch.dataproviders.- BistltchGlimpseReferenceDataProvider

<b>soupTCPSessionServiceRefid</b>	
SoupTCP Session Service reference ID	
Type:	Reference
Required:	Yes
Mutability:	Reactivation required
<b>partition</b>	
Backend partition to which an ITCH Glimpse(SoupTCP) session is connected	
Type:	Integer
Required:	Yes
Mutability:	Reactivation required

## BistOuchFileReferenceDataProvider

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of the Runtime object corresponding to this configuration object.	
Type:	String
Required:	No
Mutability:	Fixed to default value

Default value:	com.camerontec.catalys.server.adapter.bist.ouch.dataproviders.-BistOuchFileReferenceDataProvider
<b>fileName</b>	
Path to the file with static BIST Reference Data	
Type:	String
Required:	Yes
Mutability:	Reactivation required

## BreakOutBox

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.breakout.BreakOutBox
<b>persistenceFile</b>	
The base name of the files that the break out box uses to keep it's state.	

Type:	String
Required:	Yes
Mutability:	Reactivation required
<b>policyEngineClass</b>	
This specifies the service name of the market data provider service.	
Type:	String
Required:	Yes
Mutability:	Reactivation required

## BreakOutLogger

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	Yes
Mutability:	Reactivation required

# CMEiLink2MessageFactory

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.core.message.CMEiLink2MessageFactory
rawdata	
The value of the RawData field (FIX field 96) of every Logon message created by this factory.	
The RawDataLength is calculated and added to the Logon message.	
Type:	String
Required:	Yes
Mutability:	Reactivation required

# CatalysRulesEngine

id	
ID of configuration element	

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### rulesPackId

ID of rules pack this engine is using.

Type:	Reference
Required:	Yes
Mutability:	Reactivation required

### class

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.rules.CatalysRulesEngine

### msgTypes

The FIX message types to process with this listener. If not set, all message types will be processed.

Type:	String
Required:	No
Mutability:	Restart required

### persisterPath

Directory for persisting and replicating data associated with actions with a storage function.

If this directory is not specified then no persistence is performed.

If this attribute is specified, the `CatalysRulesEngine` element can have a nested `Properties` element to further configure the persisted data. This nested `Properties` element can contain [List Collection Properties](#).

The files in this directory are compacted as part of the application's end-of-day procedure, which can be [scheduled](#), or called via the [CLI](#).

Type:	String
Required:	No
Mutability:	Reactivation required

### tracking

Controls whether a message is tracked as it passes through the rules engine.

If `true`, then a compact representation of the execution path for each message is logged. The log entry shows each rule, condition and action that was executed, including the status of each. The entries are prefixed with a sequential identifier, which can be correlated with the rules configuration to determine the path of the message through the Rules Engine.

If set to `verbose`, additional tracking information is printed, including rule descriptions, condition types and action types.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false   verbose

### logging

Controls whether a message is logged as it passes through the rules engine.

If `true` then the message is logged on both entry and exit from the rules engine.

Type:	String
Required:	No



Mutability:	Reactivation required
Default value:	false
Legal values:	true   false
<b>rejectMapping</b>	
Controls whether a reject mapper is enabled.	
If true then Reject messages will be routed to the destination with the correct RefSeqNum by rules engine.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false
<b>orderStatusesToArchive</b>	
A list of order status values that indicate that an order is closed and can be archived.	
Archived orders are stored on disk, so that in-memory storage of orders does not grow without bound. If the node is re-started, this archive is deleted, and re-created from persistence files (if persistence is configured). If no statuses are specified then the following default list is used: Canceled, Expired, Done for Day, Filled, Rejected. Value NONE means no archiving. These values must match the allowable values for the OrdStatus FIX field (FIX tag 39).	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>endOfDayClearingCondition</b>	
A Rules Expression specifying a condition which determines which orders to purge from the rules message store at end of day.	

This parameter allows specification of a condition or set of conditions that determine whether a message will be cleared at the end of the day. This specification can be any expression allowed by the [Rules Engine Expression Language](#).

The following value clears orders which were only valid for the day (that is they do not have a TimeInForce field, or its value is set to zero): "isFieldPresent(TimeInForce) || (isFieldPresent(TimeInForce) && TimeInForce == 0)"

This example clears orders that have closed (that is, the value of their OrdStatus field is one of: Canceled, Filled, Expired, Done for Day, or Rejected): "matches(OrdStatus, '[42C38]')"

These expressions can also be combined logically.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	matches(OrdStatus, '[42C38]')    !isFieldPresent(TimeInForce)    TimeInForce == 0

#### **handleAmendOrCancelRequestWithNoOrder**

Controls whether Amend and Cancel requests are handled when Order request is not received.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

#### **routeToAdapterFixTag**

The fix tag added to persisted message record when routing message to the adapter.

Type:	String
Required:	No
Mutability:	Reactivation required

Default value:	7575
----------------	------

## ClearTextFilePasswordStore

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### class

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.passwordstore.- ClearTextFilePasswordStore

### passwordFile

The name of the password file.

The file format consists of one password entry per line. Password entries have the format: username:password:YYYY-MM-DD where the date is the password activation date. The current password entry for a given username is then the most recent entry with an activation date prior or equal to the current day. Lines beginning with whitespace or '#' are ignored.

Type:	String
Required:	Yes
Mutability:	Reactivation required

**lastSuccessfulPasswordFile**

The name of the file, where the password which was most recently successfully used is written.

This file format consists of one password entry per line. Password entries have the format:  
username:password.

Type:	String
Required:	No
Mutability:	Reactivation required

## ClusterNode

**id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

**dataVersionId**

Integer identifying the version of persistence data of the cluster node.

It is used during synchronization of state data to identify and resolve conflicts in the persistence files of different cluster nodes.

This value must not change across the lifetime of a persistence file. Each cluster node in the high availability cluster must have a unique dataVersionId. The value must be an integer between -127 and 128.

Type:	Integer
Required:	Yes
Mutability:	Restart required

## Configuration Reference

Minimum value:	-127
Maximum value:	128
<b>host</b>	
Host name or IP address of cluster node.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>port</b>	
The port on which the group communications of the high availability cluster occur.	
Type:	Integer
Required:	Yes
Mutability:	Restart required
Minimum value:	0
Maximum value:	65535
<b>localHost</b>	
Host name or IP address of local cluster node for outgoing connections.	
The name or IP address to locally bind connections made from this ClusterNode to all other ClusterNodes. This may be used to force HA related connections to be initiated over a specific interface rather than the default, provided the interface is configured with its own IP address.	
Type:	String
Required:	No
Mutability:	Restart required
<b>ImaPort</b>	

Port on which a Cluster Node's LMA service communicates on with other Cluster Nodes. If you change the default value of the `external-jmx-server-url` in the LMA, then this must be updated to match.

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	10002
Minimum value:	0
Maximum value:	65535

### **automaticFailover**

Controls whether the cluster node will automatically failover.

When set to 'false', this node will not automatically become the primary. It must be manually failed over using the 'cl\_primary' CLI command. Note that at least one node in the cluster must be able to automatically become the primary.

This attribute is typically disabled (i.e. set to 'false') for a disaster recovery (DR) node and/or when only explicit manual failover to this node is acceptable.

Type:	Boolean
Required:	No
Mutability:	Restart required
Default value:	true

### **processReplayRequests**

Controls whether the cluster node will process data replay requests from other nodes.

When set to 'false', this node will not process data replay requests from other nodes, which means that replicated data on this node will never propagate to any other node.

This attribute is typically disabled (i.e. set to 'false') for a disaster recovery (DR) node so that potentially stale data on this node can never propagate back to other nodes via the automatic data synchronization process.

Type:	Boolean
Required:	No
Mutability:	Restart required
Default value:	true

### **synchronizationWindow**

Used to place an upper bound on the maximum number of missed writes, per collection, for which synchronization can proceed.

When set to a value  $> 0$ , this node will not resynchronize data from other nodes when it has missed more deltas than the size of the synchronization window. When set to 0 the synchronization window is effectively unbounded.

This attribute is typically used to prevent disruption (caused by synchronization of large data sets) to a running cluster while under heavy load. Typically it is considered an operational error to start up a cluster node which is a long way behind other nodes and this attribute can be used to prevent such 'mistakes' from impacting the rest of the cluster.

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	0

### **synchronizationAttemptIntervalInitial**

Initial interval for synchronization attempt after successful synchronization and some data synchronized. Used when passive site replication is on.

This initial interval will be increased after synchronization request returns no data.

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	1000

Minimum value:	0
<b>synchronizationAttemptIntervalMax</b>	
Max interval for synchronization attempt after successful synchronization and no data synchronized. Used when passive site replication is on.	
Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	10000
Minimum value:	0
<b>synchronizationAttemptsWithSameInterval</b>	
How many synchronization attempts will be done before increasing the interval and no data synchronized. Used when passive site replication is on.	
Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	2
Minimum value:	1
<b>multicastInterfaceName</b>	
Multicast interface name	
Type:	String
Required:	No
Mutability:	Restart required



# ClusterSite

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required

# CmeMessageFactory

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.core.message.CmeMessageFactory
loginRouteID	
The value to put into the special header field (tag 9716) for every message created by this factory.	

Type:	String
Required:	Yes
Mutability:	Reactivation required

## ComplianceProcessor

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.compliance.ComplianceProcessor
msgTypes	
The FIX message types to process with this listener.	
If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required

**rulesFile**

This specifies the compliance rules file to be used. The filter set specified must be fully defined in the rules file specified.

Type:	String
Required:	Yes
Mutability:	Reactivation required

**filterSet**

This specifies the filter set id within the given compliance rules file.

Type:	String
Required:	Yes
Mutability:	Reactivation required

**marketDataServiceId**

This specifies the filter set id within the given compliance rules file.

Type:	Reference
Required:	Yes
Mutability:	Reactivation required

**priceStepTableDefinitions**

This specifies the file containing price step table definitions.

Type:	String
Required:	No
Mutability:	Reactivation required

**exposureFile**

This specifies the path of the files to be used to store exposure filter state.

## Configuration Reference

Type:	String
Required:	No
Mutability:	Reactivation required
<b>tradeExposureFile</b>	
This specifies the path of the files to be used to store trade exposure filter state.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>breakOutBoxId</b>	
The name of the BreakOut Box to use.	
Type:	Reference
Required:	No
Mutability:	Reactivation required
<b>marketMirrorId</b>	
The name of the Catalys Market Mirror to use.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>breakOutBoxAnnotationTag</b>	
The default tag added to broken out messages with annotation.	
Type:	String
Required:	No

Mutability:	Reactivation required
-------------	-----------------------

<b>updateFilterStateFromFix</b>	
<p>If set to TRUE the order counter will be updated and the decision made on incoming FIX messages rather than return messages. So, the order counter will be incremented when an NewOrderSingle message is received rather than when an ack (ExecutionReport New) from the sell side is received. For the ConcurrentOrderFilter for example, if set to FALSE, there can be more orders in the market than the actual limit if many orders are entered in quick succession before the ack's are received back</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

<b>breachIfNoTradingPrice</b>	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

<b>askBidPriceOverlapCausesPreopenFilterToStayActive</b>	
<p>In the market open phase there can still be a discrepancy in prices where the ask is greater than the bid for instance. If set to TRUE, this scenario will still be considered as PREOPEN even if the status of the instrument is set to NORMAL.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required

Default value:	false
Legal values:	true   false

## ConfigurableMessageClassMessageFactory

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.core.message.- ConfigurableMessageClassMessageFactory
messageClass	
Class name of message class to create with this message factory.	
Must implement <a href="#">IFIXMessage</a> .	
Type:	String
Required:	Yes
Mutability:	Reactivation required

Default value:	com.camerontec.catalys.server.configuration.model.- FIXMessageFactoryConfigurationObjectBase
<b>username</b>	
The value to put into the FIX Username field (FIX tag 553) of Logon messages created by this factory.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>password</b>	
The value to put into the FIX Password field (FIX tag 554) of Logon messages created by this factory.	
Type:	String
Required:	No
Mutability:	Reactivation required

## ConsolidatedDropCopy

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String

Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.dropcopy.ConsolidatedDropCopy
<b>ImaJmxServiceUrl</b>	
<p>JMX URL used to expose its management interface.</p> <p>This URL should be a well-formed JMX URL of the form: service:jmx:&lt;protocol&gt;://[&lt;host&gt;[:&lt;port&gt;]] [&lt;url-path&gt;].</p> <p>When not specified, this attribute defaults to service:jmx:jmxmp://localhost:&lt;port&gt;, where &lt;port&gt; is the port value assigned as per the range settings.</p> <p>If the value is not a well-formed JMX URL, or an error occurs while attempting to create a connector server listening on the given JMX URL (because for example, the protocol specified in the URL is not available in the current JMX implementation, or because the port is already in use) an error will be printed in the logs and the service will not be initialized, which will prevent the application from starting.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>replicationJmxPort</b>	
<p>Port on which the client replication JMX service will be run.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>fileWriteBlockSize</b>	
<p>The logger end point that will be used by the Logs replicator to connect and receive remote logs.</p>	
Type:	String
Required:	No



Mutability:	Reactivation required
<b>connectionDelayLimit</b>	
Logs replication connection delay limit	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	10000
<b>connectionManagerListenPortRangeStart</b>	
Logs replication connection manager listen port range start	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	12000
<b>connectionManagerListenPortRangeEnd</b>	
Logs replication connection manager listen port range end	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	12500
<b>connectionManagerBindAddress</b>	
Logs replication connection manager bind address	
Type:	String

## Configuration Reference

Required:	No
Mutability:	Reactivation required
Default value:	
<b>socketReceiveBufferSize</b>	
Logs replication socket receive buffer size	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	0
<b>applicationId</b>	
ID of the application we are configuring the drop copies for. This needs to be the same id configured in the xml file.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>sessionNames</b>	
Comma separated fully qualified names of the session(s) that you need the drop-copies for.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>logsRootDir</b>	
The name of the directory that will be used by the logs replicator to copy remote logs to.	
Type:	String

## Configuration Reference

Required:	No
Mutability:	Reactivation required
<b>persistenceDir</b>	
The name of the directory used to store the played logs indexes.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>clearOnReset</b>	
Indicates whether played logs position should be reset. Will replay logs from the startPoint according the configuration.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false
<b>useOriginalComplds</b>	
Indicates whether drop copy messages should retain the original session compld's	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false
<b>resumeFromPreviousPosition</b>	

Used only when 'startPoint' is SESSION. Indicates whether drop copies should be resumed from previous end point or from current/latest session start upon a disconnection

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

### startPoint

Defines the starting point from which drop copies need to be sent

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	TODAY
Legal values:	TODAY   WEEK   MONTH   YEAR   ALL   SESSION

### side

Used when start point is SESSION. Specifies the side of the session for which the drop copy session is being setup

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	sell
Legal values:	sell   buy

### msgTypes

The FIX message types to process with this listener. If not set, all message types will be processed.

Type:	String
Required:	No
Mutability:	Restart required

## CustomConnection

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	Yes
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.server.configuration.model.- ConnectionConfigurationBase
customAttributes	
Type:	Object
Required:	No
Mutability:	Reactivation required

# CustomMarketDataService

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Reactivation required
nonStandardMarketDataFields	
Non standard market data fields.	
Type:	Object
Required:	No
Mutability:	Reactivation required
customAttributes	
Type:	Object
Required:	No
Mutability:	Reactivation required

# CustomMessageFactory

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	Yes
Mutability:	Reactivation required
customAttributes	
Type:	Object
Required:	No
Mutability:	Reactivation required

# CustomMessageProcessor

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String

Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	Yes
Mutability:	Reactivation required
<b>customAttributes</b>	
Type:	Object
Required:	No
Mutability:	Reactivation required

## CustomMonitor

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	Yes



Mutability:	Reactivation required
<b>customAttributes</b>	
Type:	Object
Required:	No
Mutability:	Reactivation required

## CustomPersister

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	Yes
Mutability:	Reactivation required
<b>customAttributes</b>	
Type:	Object
Required:	No
Mutability:	Reactivation required

# CustomSelector

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	Yes
Mutability:	Reactivation required
customAttributes	
Type:	Object
Required:	No
Mutability:	Reactivation required

# CustomService

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes

Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	Yes
Mutability:	Reactivation required
<b>customAttributes</b>	
Type:	Object
Required:	No
Mutability:	Reactivation required

## DES

<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	Yes
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.core.crypto.CryptixCrypto
<b>encryptmethod</b>	
Encryption method used by this element.	
For this DES element, this attribute has the fixed value of '2'.	
Type:	String
Required:	No

Mutability:	Fixed to default value
Default value:	2

## DefaultCompression

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required

## EMXDelimitedFileAdapter

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value

Default value:	com.camerontec.catalys.server.adapter.emx.processor.- EMXDelimitedFileAdapter
<b>msgTypes</b>	
The FIX message types to process with this processor. If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required
<b>templateDirectory</b>	
The directory containing templates	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>outgoingDirectory</b>	
The directory containing outgoing messages	
The directory from which files of outgoing messages will be read and their acknowledgements from EMX will be written	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>incomingDirectory</b>	
The directory containing incoming messages	
The directory into which files of incoming messages will be written	
Type:	String

Required:	No
Mutability:	Reactivation required
<b>incomingSwitchTimer</b>	
The value of the incoming message file switch timer.	
Type:	String
Required:	No
Mutability:	Reactivation required

## EMXMessageSigner

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.emx.processor.- EMXMessageSigner
<b>msgTypes</b>	

The FIX message types to process with this processor. If not set, all message types will be processed.

Type:	String
Required:	No
Mutability:	Restart required

## EMXMessageVerifier

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### class

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.emx.processor.- EMXMessageVerifier

### msgTypes

The FIX message types to process with this processor. If not set, all message types will be processed.

Type:	String
Required:	No

Mutability:	Restart required
<b>verify</b>	
Define whether received EMX messages should be verified against our truststore or not.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	NONE
Legal values:	NONE   PARTICIPANTS   ALL

## EmbeddedCatalysNodeAdapter

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.embedded.- EmbeddedCatalysNodeAdapterProcessor



**msgTypes**

The FIX message types to process. If not set, all message types are processed.

Type:	String
Required:	No
Mutability:	Restart required

## EncryptedFilePasswordStore

**id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

**class**

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.passwordstore.- EncryptedFilePasswordStore

**passwordFile**

The name of the password file.

The file format consists of one password entry per line. Password entries have the format: username:password:YYYY-MM-DD where the date is the password activation date. The current

password entry for a given username is then the most recent entry with an activation date prior or equal to the current day. Lines beginning with whitespace or '#' are ignored.

Type:	String
Required:	Yes
Mutability:	Reactivation required

### **lastSuccessfulPasswordFile**

The name of the file, where the password which was most recently successfully used is written.

This file format consists of one password entry per line. Password entries have the format: username:password.

Type:	String
Required:	No
Mutability:	Reactivation required

## EncryptionSource

### **id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### **class**

Java class of this element's Runtime object.

Type:	String
Required:	No

Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.test.EncryptionSource
<b>encrypt.fields</b>	
A comma delimited list of tag numbers of fields to be encrypted.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>encrypt.all.possible.fields</b>	
Whether or not to encrypt all fields that can be encrypted.	
Fields that will not be encrypted are: BeginString, BodyLength, CheckSum, MsgType, SecureDataLen, SecureData, SignatureLength, Signature.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false
<b>encrypt.recommended.fields</b>	
Whether or not to always encrypt the fields recommended by fixprotocol.org).	
The recommended fields are: SendingTime, MsgSeqNum, PossDupFlag, SenderCompID, TargetCompID.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true

Legal values:	true   false
---------------	--------------

## Event

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
task	
Refers to a defined task by its ID.	
Type:	Reference
Required:	No
Mutability:	Reactivation required
schedule	
Refers to a defined schedule by its ID.	
Type:	Reference
Required:	No
Mutability:	Reactivation required

## EventList

id
ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### parentEventList

A reference to another event list.

Can be used to include a common list of events.

Type:	Reference
Required:	No
Mutability:	Reactivation required

## FASTCompression

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### templateFile

Name of file containing FAST templates in XML format.

Type:	String
Required:	No
Mutability:	Reactivation required

**blocking**

Controls whether blocks of messages are used or messages are sent one at a time.

See the FAST 1.1 specification available on the FIX protocol website, section 10: Transfer Encoding - "A FAST stream consists of a sequence of messages or a sequence of blocks. A block has a leading block size specifying the number of bytes occupied by the messages of the block."

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

**overlongBlocksizeLength**

The length of the overlong integer representing the block size (in bytes) after encoding.

This attribute is only taken into consideration if blocking is enabled.

When blocking is enabled the block size is represented as an Unsigned Integer which may be 'overlong'. An integer is 'overlong' if the entity value still represents the same integer after removing seven or more of the most significant bits. This parameter defines the length of the overlong integer after encoding. See the FAST 1.1 specification available on the FIX protocol website, section 10.6.1, for more details.

If set to zero (the default), a standard integer is used for the block size.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	0
Minimum value:	0

**clearPolicy**

The dictionary clearing strategy to use.

This must be equal to 'always' (for clearing dictionaries on every message) or 'never'. The clearing strategy is something both sides of the connection need to agree on. It also depends on the reliability

of the transport. If messages can be lost or come in out of order, then 'always' is the only practical option. If the transport is reliable 'never' will provide significant performance improvements.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	always
Legal values:	always   never

### ScpSupport

Level of FAST Session Control Protocol (SCP) support provided by the server.

It is 'SCP11L1' for SCP 1.1 level 1 or 'SCP11L2' for SCP 1.1 level 2. If not specified, then no SCP support is provided.

Type:	String
Required:	No
Mutability:	Reactivation required
Legal values:	SCP11L1   SCP11L2

## FastFriendlyFIXMessageFactory

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### class

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.core.message.FastFriendlyFIXMessageFactory

### templateFile

The name of the file containing the FAST templates.

Type:	String
Required:	Yes
Mutability:	Reactivation required

## FieldFilterByFixVersion

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### class

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value



Default value:	com.camerontec.catalys.server.processor.FieldFilterByFixVersion
<b>msgTypes</b>	
The FIX message types to process. If not set, all message types are processed.	
Type:	String
Required:	No
Mutability:	Restart required

## FileLookupService

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.lookup.FileLookupService
<b>fileName</b>	
The full path to a delimited text file.	
Type:	String

## Configuration Reference

Required:	Yes
Mutability:	Reactivation required
<b>delimiter</b>	
A single character that delimits columns in the data file.	
Type:	String
Required:	Yes
Mutability:	Reactivation required
<b>skipHeaderLines</b>	
The number of lines to skip at the beginning of the file.	
If the data file contains column names or other header lines, they can be skipped by setting this attribute.	
Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	0
Minimum value:	0
Maximum value:	65535
<b>commentCharacter</b>	
Lines in the lookup file beginning with this character are ignored. Default is #	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	#

**lookupKeys**

A comma-separated list of lookup key definitions.

A lookup key is made of one or more integers that correspond to columns in the data file. If the key contains multiple columns, the value must be wrapped in braces, e.g. {1,2}. Multiple lookup keys can be defined in order to support different ActionFileLookup rules that refer to different columns. For example: 0,{1,2,3},4

Type:	String
Required:	Yes
Mutability:	Reactivation required

**lookupKeyLabels**

A comma-separated list of names that corresponds to each lookup key.

These are used only for display in the Dashboard.

Type:	String
Required:	No
Mutability:	Reactivation required

**resultIndexes**

A comma-separated list of columns that contain result values.

Type:	String
Required:	Yes
Mutability:	Reactivation required

**resultIndexLabels**

A comma-separated list of names that corresponds to each result index.

These are used only for display in the Dashboard.

Type:	String
-------	--------

Required:	No
Mutability:	Reactivation required

## FixLoginAuthenticationMonitor

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.security.auth.login.- FixLoginAuthenticationMonitor
configName	
The name of the login module configuration to be used by this monitor.	
Type:	String
Required:	Yes
Mutability:	Reactivation required
monitorInterval	

The interval between successive checks of FIX session status in milliseconds.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Minimum value:	0

## FixMarketMirror

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### class

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.core.marketstate.FixMarketMirror

### defaultFIXVersion

Default version of FIX protocol.

Type:	String
Required:	No

Mutability:	Reactivation required
Legal values:	4.0   4.1   4.2   4.3   4.4   5.0   5.0SP1   5.0SP2

## FixMarketMirrorWithOrderArchive

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.core.marketstate.- FixMarketMirrorWithOrderArchive
orderStatusesToArchive	
A list of values of order status that indicate that an order is closed and can be archived.	
Only used if the market mirror populator is capable of archiving. This is the case when the marketMirrorClass attribute has been set to <a href="#">FixMarketMirrorWithOrderArchive</a> or a class that implements <a href="#">IMarketMirrorWithOrderArchive</a> .	
If no statuses are specified then the following default list is used: Canceled, Expired, Done for Day, Filled, Rejected.	
Type:	String

Required:	No
Mutability:	Reactivation required
<b>archiveFileName</b>	
<p>Name of archive log.</p> <p>Only used if the market mirror populator is capable of archiving. This is the case when the marketMirrorClass attribute has been set to <a href="#">FixMarketMirrorWithOrderArchive</a> or a class that implements <a href="#">IMarketMirrorWithOrderArchive</a>.</p> <p>If no statuses are specified then the following default list is used: Canceled, Expired, Done for Day, Filled, Rejected.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>archiveImplClass</b>	
<p>Market mirror order archive class name.</p> <p>Only used if the market mirror populator is capable of archiving. This is the case when the marketMirrorClass attribute has been set to <a href="#">FixMarketMirrorWithOrderArchive</a> or a class that implements <a href="#">IMarketMirrorWithOrderArchive</a>.</p> <p>If no statuses are specified then the following default list is used: Canceled, Expired, Done for Day, Filled, Rejected.</p> <p>This class must implement the <a href="#">IOrderArchive</a> interface.</p> <p>If not specified <a href="#">OrderArchiveLog</a> will be used.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>defaultFIXVersion</b>	
Default version of FIX protocol.	

Type:	String
Required:	No
Mutability:	Reactivation required
Legal values:	4.0   4.1   4.2   4.3   4.4   5.0   5.0SP1   5.0SP2
<b>dictionaryName</b>	
Dictionary to use for creating of Order Record.	
It should be supplied for OrderArchiveLog created.	
Type:	String
Required:	No
Mutability:	Reactivation required

## FlatFileAdapter

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value



Default value:	com.camerontec.catalys.server.adapter.flatfile.FlatFileAdapter
<b>msgTypes</b>	
The FIX message types to process with this listener. If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required
<b>peerApplicationManagerService</b>	
The ID of a PeerApplicationManagerService.	
This needs to be specified when the configuration contains multiple sessions that read from the same file. The PeerApplicationManagerService handles the dispatching of FIX messages to the appropriate FIX session based on the FIX addressing fields contained in the message.	
Type:	Reference
Required:	No
Mutability:	Restart required
<b>autoForwarding</b>	
Controls whether messages are automatically forwarded to other listeners in the chain.	
If true, messages are automatically forwarded.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false
<b>peerWriter</b>	
Class that writes FIX messages into the flat file.	

This class must implement [IPeerApplicationWriter](#).

Type:	String
Required:	No
Mutability:	Restart required
Default value:	com.camerontec.catalys.server.adapter.flatfile.DefaultFlatFileWriter

### fromFixQueue

The name of the directory (relative to where the application is running) in which to write entries.

Type:	String
Required:	Yes
Mutability:	Reactivation required

### peerReader

Class that reads FIX messages entries from the flat file.

This class must implement [IPeerApplicationReader](#).

Type:	String
Required:	No
Mutability:	Restart required
Default value:	com.camerontec.catalys.server.adapter.flatfile.DefaultFlatFileReader

### peerRecoveryInterval

The recovery interval (in milliseconds) used by the reader after a queue reader error.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	10

**peerPollingInterval**

The polling interval (in milliseconds) used by the reader when waiting for files to read.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	10

**toFixQueue**

The name of the directory (relative to where the application is running) from which to read entries.

Type:	String
Required:	Yes
Mutability:	Reactivation required

**toFixErrorQueue**

The name of the directory (relative to where the application is running) to write error entries.

These entries are those that have been read from the "toFixQueue" but cannot be sent.

Type:	String
Required:	No
Mutability:	Reactivation required

**transformer**

The transformer class that implements IMessageTransformer to convert inbound/outbound FIX messages and events.

Type:	String
Required:	No
Mutability:	Restart required

Default value:

com.camerontec.catalys.core.message.StandardFixTransformer

## GenericBecomingPrimaryAction

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of the Runtime object which implements the action.	
This class must implement <a href="#">IGenericBecomingPrimaryAction</a> , whose execute method is used to carry out the custom action.	
Type:	String
Required:	Yes
Mutability:	Restart required

## GenericMessageFactory

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required

class	
Java class of this element's Runtime object.	
Type:	String
Required:	Yes
Mutability:	Reactivation required

## GenericMonitor

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	Yes
Mutability:	Reactivation required

## GenericPersister

id	
ID of configuration element	
Must be unique across the same configuration element types.	

Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	Yes
Mutability:	Reactivation required

## GenericPrimaryValidityCheck

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
The Java class which implements the validity check.	
This class must implement <a href="#">IPrimaryValidityCheck</a> , whose performCheck method is used to carry out the custom check.	
Type:	String
Required:	Yes
Mutability:	Restart required

## GenericProcessor

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	Yes
Mutability:	Reactivation required
msgTypes	
The FIX message types to process with this processor. If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required

## GenericRelinquishingPrimaryAction

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String

Required:	Yes
Mutability:	Restart required
<b>class</b>	
The Java class which implements the relinquishing primary action.	
This class must implement <a href="#">IGenericRelinquishingPrimaryAction</a> , whose execute method is used to carry out the custom action.	
Type:	String
Required:	Yes
Mutability:	Restart required

## GenericSelector

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	Yes
Mutability:	Reactivation required



## GenericService

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	Yes
Mutability:	Reactivation required

## HighAvailabilityCluster

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
groupPublishPeriod	
Specifies the frequency of group updates in milliseconds.	
Type:	Integer

## Configuration Reference

Required:	No
Mutability:	Restart required
Default value:	1000
Minimum value:	1
<b>groupAgeThreshold</b>	
The number of 'groupPublishPeriod' intervals that the group will wait for an update from a node in the group before removing it from the group.	
Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	3
Minimum value:	3
<b>primaryValidityChecksTimeout</b>	
The maximum amount of time (in milliseconds) that a node is allocated to perform its primary validity checks.	
If this time is exceeded then the node cannot become the primary.	
Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	10000
Minimum value:	1
<b>becomingPrimaryActionsTimeout</b>	
The maximum amount of time (in milliseconds) that a node is allowed to perform its becoming primary actions.	

If this time is exceeded then the node cannot become primary and it leaves the group.

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	10000
Minimum value:	1

### **relinquishingPrimaryActionsTimeout**

The maximum amount of time (in milliseconds) that a node is allowed to perform its relinquishing primary actions.

If this time is exceeded then the node leaves the group.

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	10000
Minimum value:	1

### **replicationTimeout**

The maximum amount of time (in milliseconds) given to the primary for replication of its state. If this time is exceeded, the primary proceeds, but the other nodes in the group will need to resynchronize with the primary.

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	3000
Minimum value:	1

**becomePrimaryWithoutLMAConfirmation**

Controls whether the node can become primary without LMA confirmation.

If set to 'false' then before becoming primary, a node will confirm (using the LMA) that the previous primary application is not available. If the LMA cannot confirm this, or the LMA cannot be contacted, then the node will not be able to become primary.

Type:	Boolean
Required:	No
Mutability:	Restart required
Default value:	true

**disableReplication**

Controls whether the state of the primary is replicated on the other nodes of the group.

When set to 'true', the state is not replicated. This setting is to be used in networks which perform their own replication e.g. SAN environments.

Type:	Boolean
Required:	No
Mutability:	Restart required
Default value:	false

**disableSoftwareRevisionValidation**

Controls whether software revisions of cluster nodes are compared when the node starts up.

When set to 'true', software revisions are not compared. When set to 'false', software revisions are compared and the cluster node will not start if there is a revision mismatch.

Type:	Boolean
Required:	No
Mutability:	Restart required
Default value:	false

**maxClusterMessageSize**

The maximum size of any cluster message in bytes.

If this value is specified then it must be greater than or equal to 1024.

By default all outgoing FIX messages are replicated and therefore this limit must be set large enough to accommodate the largest FIX message. The serialized representation of any single persisted collection must always be smaller than this limit since writes to persisted collections are also replicated.

*It is not possible to configure the system to adapt to arbitrarily large cluster messages. Since replication requests which cannot fit into the maximum cluster message are rejected it is critical to understand and size this attribute correctly. Only in very rare circumstances will a single FIX message cause the default limit to be exceeded. Rather, very large write requests are typically the result of programmatically using the `Catalys Persistent Collections` framework to persist large (when serialized) values. Often this indicates a programming error and/or a bad serialization mechanism (e.g. inadvertent default Java serialization of a large object graph).*

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	65536

**tcpNoDelay**

Controls whether Nagle's algorithm is used by HA connections.

When true, Nagle's algorithm is disabled and the TCP/IP stack uses the TCP\_NO\_DELAY option. By default Nagle's algorithm is used.

Type:	Boolean
Required:	No
Mutability:	Restart required
Default value:	false

**tcpKeepAlive**

Controls whether keep alive is enabled on HA connections.

When true, the socket keep alive will be turned on. By default the keep alive option is disabled.

Type:	Boolean
Required:	No
Mutability:	Restart required
Default value:	false

### **soLinger**

Controls whether the HA communications sockets should linger on close.

When set to non-negative values, the socket linger timeout will be set. By default, it is disabled

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	-1

### **tcpSendBufferSize**

TCP socket send buffer size

Size of the TCP socket send buffer requested from the OS. The node will not start unless it can obtain a buffer of the required size.

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	0

### **tcpReceiveBufferSize**

TCP socket receive buffer size

Size of the TCP socket receive buffer requested from the OS. The node will not start unless it can obtain a buffer of the required size.

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	0

**allowConfigMismatch**

Controls whether the node will start in the presence of configuration differences across the cluster.

If this node's configuration is different from the other nodes in the cluster then by default the node will not start. Setting this option to true will allow the node to start in the presence of configuration differences.

Enabling this option is not recommended, as cluster nodes should have identical configurations so that they are interchangeable when a failover occurs.

*Note: This does not apply to the HighAvailabilityCluster configuration itself, which must be identical across the cluster for the node to start up.*

Type:	Boolean
Required:	No
Mutability:	Restart required
Default value:	false

**passiveSiteReplication**

Controls whether the passive site replication is on.

When set to 'true' all nodes in sites other than the site in which the primary is running periodically poll a secondary node to synchronize their data rather than having the primary push updates to them. The polling periodicity is based on the synchronization interval attributes of the ClusterNode.

*If there are no active secondary nodes in the current primary's ClusterSite, then nodes from other sites will begin to poll the primary node until a secondary returns.*

This option is required when multicast based replication is enabled but datagrams do not propagate outside the primary data site.

Type:	Boolean
-------	---------

Required:	No
Mutability:	Restart required
Default value:	false
<b>multicastAddress</b>	
Data replication multicast address	
<p>When a multicast address and port are supplied then the cluster attempts to replicate data using multicast. Note that only data which is likely not to be fragmented (as defined by the 'maxDatagramSize' attribute) is sent via multicast. Larger write requests are sent via TCP.</p> <p>Format depends on the 'multicastProtocolFamily' which defaults to 'INET' (IPv4).</p> <p><i>Note that when multicast based replication is enabled it is a fundamental consideration whether UDP datagrams are received by Nodes at all sites. This is a characteristic of the underlying network and cannot be modified by the HA software. Unless all nodes are capable of receiving UDP datagrams from the primary node, then 'passiveSiteReplication' mode must also be enabled to keep nodes in non-multicast sites synchronized.</i></p> <p>Note that it may be necessary to set a non default value for 'multicastTimeToLive' in order for UDP datagrams to reach all nodes.</p>	
Type:	String
Required:	No
Mutability:	Restart required
<b>multicastPort</b>	
Data replication multicast port	
See the 'multicastAddress' attribute for more details.	
Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	0



**multicastProtocolFamily**

Data replication multicast protocol family. Only relevant when multicast replication is enabled.

Use 'INET6' (instead of the default 'INET') to use an IPv6 'multicastAddress'.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	INET

**multicastTimeToLive**

Multicast packet TTL (time to live). Only relevant when multicast replication is enabled.

For 'INET' (IPv4) datagram sockets this specifies the multicast TTL. For 'INET6' (IPv6) datagram sockets this specifies the maximum hop count.

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	1

**maxDatagramSize**

Max multicast datagram size. Only relevant when multicast replication is enabled.

Limits the size of datagram packets that will be sent over multicast.

Limiting this size (as opposed to using the MTU of the network interface) increases the probability that datagram packets reach their destination without being fragmented. It is recommended to set this value to the measured path MTU between cluster nodes.

If/when an attempt is made to send a replication packet which would exceed this length it is sent via TCP instead of multicast.

Type:	Integer
Required:	No

Mutability:	Restart required
Default value:	576
<b>udpSendBufferSize</b>	
<p>UDP datagram socket send buffer size. Only relevant when multicast replication is enabled. The node will not start unless it can obtain a buffer of the required size.</p> <p>Size of the datagram socket send buffer requested from the OS.</p>	
Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	0
<b>udpReceiveBufferSize</b>	
<p>UDP datagram socket receive buffer size</p> <p>Size of the datagram socket receive buffer requested from the OS. The node will not start unless it can obtain a buffer of the required size.</p> <p>When a datagram arrives at a node (e.g. a multicast write request) it is silently discarded unless the datagram socket's receive buffer has enough spare capacity to accommodate the datagram. This is the most common cause of lost datagram packets, therefore in a high throughput system it is recommended that this value be set as large as possible. It may be necessary to tune the host networking stack to allow large values.</p>	
Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	0
<b>minimizeContextSwitchLatency</b>	
Prefer minimal context switch latency over lower CPU load	

True to attempt to minimize context switch latency at the cost of higher CPU load. Concretely, this attribute changes two aspects of HA communications subsystem behavior:

1. Enables pinning of hot HA threads to cores (if also enabled at the application level)
2. Sets busy/wait strategies for hot HA threads so they are less likely to incur delays after quiet periods.

Type:	Boolean
Required:	No
Mutability:	Restart required
Default value:	false

#### **outboundDataMessageBufferSize**

Outbound write request buffer size

Application level outbound write request buffer size. If value is not a power of 2, it will choose the next power of 2 value.

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	1024

#### **inboundDataMessageBufferSize**

Inbound write request buffer size

Application level inbound write request buffer size. If value is not a power of 2, it will choose the next power of 2 value.

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	131072

**collectionRegistryPath**

Path to the 'CollectionRegistry' directory where collection registry files are stored.

Path to the 'CollectionRegistry' directory where Catalys stores information about the collections that are replicated with other nodes. If this path is not absolute, it is relative to the application's working directory. This attribute is used only if replication is enabled.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	

**homogeneousEndianness**

Flag indicating whether or not all cluster nodes are running on hardware platforms with matching endianness

If any single cluster node is running on a hardware platform with a different endianness than any other cluster node this flag must be disabled (i.e. set to false) to guarantee correct communication between cluster nodes. Disabling this flag is also permissible (though potentially slightly less performant) when all nodes are running on identical hardware platforms.

Type:	Boolean
Required:	No
Mutability:	Restart required
Default value:	true

**supportsUnalignedTypeAccess**

Flag indicating whether or not all cluster nodes support unaligned type access

If any node in the cluster is running on a hardware platform that does not support unaligned type access then this flag must be disabled (i.e. set to false). x86 based systems typically supported unaligned type access, whereas risc (e.g. sparc) based systems do not. Disabling this flag is also permissible (though potentially slightly less performant) when all nodes are running on the same hardware platform.

Type:	Boolean
Required:	No
Mutability:	Restart required
Default value:	true

## HotSpotFXIMessageFactory

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.core.message.HotSpotFXIMessageFactory
messageClass	
Class name of message class to create with this message factory.	
Must implement <a href="#">IFIXMessage</a>	
Type:	String
Required:	Yes

Mutability:	Reactivation required
<b>username</b>	
The value to put into the FIX Username field (FIX tag 553) of Logon messages created by this factory.	
Type:	String
Required:	Yes
Mutability:	Reactivation required
<b>password</b>	
The value to put into the FIX Password field (FIX tag 554) of Logon messages created by this factory.	
Type:	String
Required:	Yes
Mutability:	Reactivation required
<b>onBehalfOfSubID</b>	
The value to put into the FIX OnBehalfOfSubID (FIX tag 116) of administrative messages created by this factory.	
Type:	String
Required:	Yes
Mutability:	Reactivation required

## IBMMQMessageListener

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String

Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.mq.MQProducer
<b>msgTypes</b>	
The FIX message types to process with this listener. If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required
<b>error.queue.name</b>	
The queue into which all exceptions will be placed after they have been transformed using "transformer"	
If not specified, exceptions will be placed into "queue.name" along with the other messages.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>message.format</b>	
Specifies the outgoing message format.	
This attribute should be set when a custom transformer is not using the default MQSTR format. (Use MQSTR for MQC.MQFMT_STRING for instance).	

Type:	String
Required:	No
Mutability:	Reactivation required
<b>message.charset</b>	
<p>The numerical value (positive integer) of the character set that outgoing messages use.</p> <p>This attribute should be set when a custom transformer is using a character set different from the character set of the platform on which the Catalys Server is running. (850 for Windows platform, 37 for AS400 platform for instance).</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>message.persistence</b>	
<p>The numerical value(positive integer) to set the outgoing message persistence.</p> <p>This attribute should be set when a non default persistence is required. Should be one of the MQMessage.persistence values. Default value for String data is MQC.MQPER_NOT_PERSISTENT(value 0) and for byte[] and ByteSegment it's MQC.MQPER_PERSISTENCE_AS_Q_DEF(value 2)</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>open.options</b>	
<p>The MQC MQOO options to use when opening an MQ queue.</p> <p>Options should appear as a comma-separated list of option names.</p>	
Type:	String
Required:	No



Mutability:	Reactivation required
Default value:	MQOO_OUTPUT,MQOO_FAIL_IF_QUIESCING
<b>queue.manager.name</b>	
The queue manager managing the inbound or outbound queue being used.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>queue.name</b>	
The queue to use to write or read messages respectively.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>server.ccsid</b>	
The MQ environment charset.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>server.hostname</b>	
The IP address or host name of the host on which the MQSeries Server is running.	
Only applicable for 'client' connection types.	
Type:	String
Required:	No

Mutability:	Reactivation required
<b>server.port.number</b>	
The port number of the MQSeries Server.	
Only applicable for 'client' connection types.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>server.channel.name</b>	
The name of a channel of type 'Server Connection' configured and running on the MQSeries Server.	
Only applicable for 'client' connection types, and only required if "server.hostname" is specified.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>server.userid</b>	
User identity associated with the connection to the MQSeries Server.	
Only applicable for 'client' connection types, and only required if this has been configured on the MQSeries Server.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>server.password</b>	
Password associated with the connection to the MQSeries Server.	
Only applicable for 'client' connection types, and only required if this has been configured on the MQSeries Server.	

Type:	String
Required:	No
Mutability:	Reactivation required
<b>reconnectDelay</b>	
Reconnection delay.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	10000
<b>useUnitOfWork</b>	
Controls whether unit of work is used when reading and writing messages in the queue.	
<p>If set to true, and the operation succeeds then all the message gets and puts that have occurred since the last syncpoint are to be made permanent. Messages sent as part of a unit of work (with the MQC.MQPMO_SYNCPOINT flag set in MQPutMessageOptions.options) are made available to other applications. Messages retrieved as part of a unit of work (with the MQC.MQGMO_SYNCPOINT flag set in MQGetMessageOptions.options) are deleted.</p> <p>If set to true, and the operation fails then all the message gets and puts that have occurred since the last syncpoint are to be backed out. Messages sent as part of a unit of work (with the MQC.MQPMO_SYNCPOINT flag set in the options field of MQPutMessageOptions) are deleted. Messages retrieved as part of a unit of work (with the MQC.MQGMO_SYNCPOINT flag set in MQGetMessageOptions.options) are reinstated on the queue.</p> <p>Note that if the transformer is unable to parse the message and throws an exception, an error is logged, and the message is not reinstated back on the queue, as this can cause a processing loop.</p> <p>If set to false, then no commit/backout is executed in case of success/failure.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required

Default value:	true
Legal values:	true   false
<b>transformer</b>	
<p>Class which transforms FIX messages into IBM MQSeries messages.</p> <p>Must implement <a href="#">IMessageTransformer</a>.</p> <p>The com.camerontec.catalys.server.adapter.mq.OlympicRawTransformer can be used when the MQAdapter is connected to an Olympic system.</p> <p>Note that this transformer does not perform any CSID (Character Set ID used by the MQ Series system) conversion. A customized transformer may be required to perform the character set conversion.</p> <p>Note also that since this adapter sets the storeMessages attribute to false to enable message reuse, then transformers which are intended to be used with this adapter may not store any references to the FIX message object.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.server.adapter.mq.MQTransformer
<b>sslCipherSuite</b>	
<p>CipherSuite to use when communicating with the MQ queue manager using SSL.</p> <p>This value must correspond to the cipher suite value configured on the MQSeries Server. Only applicable for 'client' connection types, and only required if this has been configured on the MQSeries Server.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>logMessages</b>	

Enables the logging of FIX messages.

If set to true, all FIX messages written to MQ will be logged at the INFO level. Defaults to false.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false

### **ccdtURL**

The URL which specifies the channel definition file to be used in connecting to the queue manager.

If configured, then you don't need to specify other server connection properties such as `server.hostname`, `server.port.number`, `server.channel.name`.

Type:	String
Required:	No
Mutability:	Reactivation required

### **autoForwarding**

Controls whether messages are automatically forwarded to other listeners in the chain.

If true, messages are automatically forwarded.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

## **IBMMQMessageSource**

**id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### **class**

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.mq.MQConsumer

### **error.queue.name**

The queue into which all exceptions will be placed after they have been transformed using "transformer"

If not specified, exceptions are logged and not placed in a queue.

Type:	String
Required:	No
Mutability:	Reactivation required

### **initial.wait**

The interval to wait for an MQ connection before timing out.

Type:	String
Required:	No
Mutability:	Reactivation required

Default value:	5000
<b>wait.interval</b>	
The interval to wait for an MQ message before timing out and trying again.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	1000
<b>open.options</b>	
The MQC MQOO options to use when opening an MQ queue.	
Options should appear as a comma-separated list of option names.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	MQOO_INPUT_AS_Q_DEF,MQOO_OUTPUT,- MQOO_FAIL_IF QUIESCING
<b>queue.manager.name</b>	
The queue manager managing the inbound or outbound queue being used.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>queue.name</b>	
The queue to use to write or read messages respectively.	
Type:	String

## Configuration Reference

Required:	No
Mutability:	Reactivation required
<b>server.ccsid</b>	
The MQ environment charset.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>server.hostname</b>	
The IP address or host name of the host on which the MQSeries Server is running.	
Only applicable for 'client' connection types.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>server.port.number</b>	
The port number of the MQSeries Server.	
Only applicable for 'client' connection types.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>server.channel.name</b>	
The name of a channel of type 'Server Connection' configured and running on the MQSeries Server.	
Only applicable for 'client' connection types, and only required if "server.hostname" is specified.	



Type:	String
Required:	No
Mutability:	Reactivation required
<b>server.userid</b>	
<p>User identity associated with the connection to the MQSeries Server.</p> <p>Only applicable for 'client' connection types, and only required if this has been configured on the MQSeries server.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>server.password</b>	
<p>Password associated with the connection to the MQSeries Server.</p> <p>Only applicable for 'client' connection types, and only required if this has been configured on the MQSeries server.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>reconnectDelay</b>	
Reconnection delay.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	10000

**useUnitOfWork**

Controls whether unit of work is used when reading and writing messages in the queue.

If set to true, and the operation succeeds then all the message gets and puts that have occurred since the last syncpoint are to be made permanent. Messages sent as part of a unit of work (with the MQC.MQPMO\_SYNCPOINT flag set in MQPutMessageOptions.options) are made available to other applications. Messages retrieved as part of a unit of work (with the MQC.MQGMO\_SYNCPOINT flag set in MQGetMessageOptions.options) are deleted.

If set to true, and the operation fails then all the message gets and puts that have occurred since the last syncpoint are to be backed out. Messages sent as part of a unit of work (with the MQC.MQPMO\_SYNCPOINT flag set in the options field of MQPutMessageOptions) are deleted. Messages retrieved as part of a unit of work (with the MQC.MQGMO\_SYNCPOINT flag set in MQGetMessageOptions.options) are reinstated on the queue.

Note that if the transformer is unable to parse the message and throws an exception, an error is logged, and the message is not reinstated back on the queue, as this can cause a processing loop.

If set to false, then no commit/backout is executed in case of success/failure.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

**transformer**

Class which transforms IBM MQSeries messages into FIX messages.

Must implement [IMessageTransformer](#).

The com.camerontec.catalys.server.adapter.mq.OlympicRawTransformer can be used when the MQAdapter is connected to an Olympic system.

Note that this transformer does not perform any CSID (Character Set ID used by the MQ Series system) conversion. A customized transformer may be required to perform the character set conversion.

Note also that since this adapter sets the storeMessages attribute to false to enable message reuse, then transformers which are intended to be used with this adapter may not store any references to the FIX message object.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.server.adapter.mq.MQTransformer

### sslCipherSuite

CipherSuite to use when communicating with the MQ queue manager using SSL.

>This value must correspond to the cipher suite value configured on the MQSeries Server. Only applicable for 'client' connection types, and only required if this has been configured on the MQSeries server.

Type:	String
Required:	No
Mutability:	Reactivation required

### backoutThreshold

The backout count threshold.

The number of times that the message can be backed out before it is committed and transferred to the backout queue specified by the backoutQueue attribute.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	-1

### backoutQueue

The name of the backout queue.

The name of the queue to which a message is transferred if it is backed out more than the number of times specified in the `backoutThreshold` attribute.

Type:	String
Required:	No
Mutability:	Reactivation required

### **logMessages**

Enables the logging of FIX messages.

If set to true, all FIX messages read from MQ will be logged at the INFO level. Defaults to false.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false

### **writeUnparsableMessagesToBackoutQueue**

Enables the writing of messages that couldn't be transformed to the backout queue.

If set to true, messages that couldn't be transformed will be written to the configured backout queue.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false

### **ccdtURL**

The URL which specifies the channel definition file to be used in connecting to the queue manager.

If configured, then you don't need to specify other server connection properties such as `server.hostname`, `server.port.number`, `server.channel.name`.

Type:	String
-------	--------

Required:	No
Mutability:	Reactivation required
<b>shouldWaitForAllSessionsToBeLoggedOn</b>	
<p>decides whether to wait for the sessions to be logged on before processing MQ messages</p> <p>If configured to true, incoming MQ messages will only be processed if the corresponding session is logged on. Otherwise, it would wait</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
Legal values:	true   false

## InternalMarketDataService

<b>id</b>	
<p>ID of configuration element</p> <p>Must be unique across the same configuration element types.</p>	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
<p>Java class of this element's Runtime object.</p>	
Type:	String
Required:	No
Mutability:	Fixed to default value

Default value:	com.camerontec.catalys.server.marketdata.service.- InternalMarketDataService
<b>nonStandardMarketDataFields</b>	
Non standard market data fields.	
Type:	Object
Required:	No
Mutability:	Reactivation required

## ItchGlimpseReferenceDataProvider

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of the Runtime object corresponding to this configuration object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.ouch.dataproviders.- ItchGlimpseReferenceDataProvider
<b>soupTCPSessionServiceRefid</b>	
SoupTCP Session Service reference ID	

Type:	Reference
Required:	Yes
Mutability:	Reactivation required
<b>partition</b>	
Backend partition to which an ITCH Glimpse(SoupTCP) session is connected	
Type:	Integer
Required:	Yes
Mutability:	Reactivation required

## JMXManagementService

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.management.JMXManagementService
<b>enabled</b>	

Controls whether the JMX Management Service is enabled or disabled.

If set to 'false', the JMX management service will not be activated.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	true

### **jmxServiceURL**

JMX URL used to expose its management interface to the LMA.

This URL should be a well-formed JMX URL of the form: service:jmx:<protocol>://[<host>[:<port>]]/[<url-path>]. When not specified, this attribute defaults to service:jmx:jmxmp://localhost:<port>, where <port> is the port value assigned as per the range settings.

If the value is not a well-formed JMX URL, or an error occurs while attempting to create a connector server listening on the given JMX URL (because for example, the protocol specified in the URL is not available in the current JMX implementation, or because the port is already in use) an error will be printed in the logs and the service will not be initialized, which will prevent the application from starting.

Type:	String
Required:	No
Mutability:	Reactivation required

### **jmxmpPortRangeStart**

Start of port range to use for assigning a port number to the JMXMP connector server.

This attribute is only taken into account when attribute 'jmxServiceURL' is not used. This attribute's value must be a positive integer, and it must be less than the value of attribute 'jmxmpPortRangeEnd'. If any of these constraints is not fulfilled, an error will be printed in the logs and the service will not be initialized, which will prevent the application from starting.

Type:	Integer
Required:	No
Mutability:	Reactivation required



Default value:	11000
Minimum value:	1
Maximum value:	65535

<b>jmxmpPortRangeEnd</b>	
End of port range to use for assigning a port number to the JMXMP connector server.	
This attribute is only taken into account when attribute 'jmxServiceURL' is not used. This attribute's value must be a positive integer, and it must be greater than the value of attribute 'jmxmpPortRangeStart'. If any of these constraints is not fulfilled, an error will be printed in the logs and the service will not be initialized, which will prevent the application from starting.	
Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	65535
Minimum value:	1
Maximum value:	65535

<b>lmaJMXServiceURL</b>	
JMX URL used to connect to the LMA.	
This attribute should only be used in situations where the LMA is using non-standard settings. This URL should be a well-formed JMX URL of the form: service:jmx:<protocol>://[<host>[:<port>]][/<url-path>].	
If the value is not a well-formed JMX URL, or an error occurs while attempting to create a client connection to the given JMX URL, an error will be printed in the logs, and the service will not be initialized, which will prevent the application from starting.	
Type:	String
Required:	No
Mutability:	Reactivation required

**applicationStartupScript**

The location of the current Catalys Server instance's startup script.

This location is passed to the LMA during the registration process, so that the LMA can later restart this instance. This is one of the two methods which can be used to specify the location of the start-up script. The other method consists in specifying the '-DJMXManagementService.startupScript' system property when starting the JVM. When the two methods are used simultaneously, the latter takes the precedence. Using a system property is the recommended method because it is easy to generate its value dynamically in a shell or bat script, whereas using a configuration property is less portable.

Type:	String
Required:	No
Mutability:	Reactivation required

**alwaysKeepJMXConnectorServerRunning**

Controls whether the JMX connector server is kept running between successive attempts at LMA registration.

The default behaviour is to start a JMX connector server just before registering with the LMA and stop it immediately if the registration fails. This is to avoid consuming resources when no clients are making use of the JMX connector server. There are situations, however, where it may be desirable to expose MBeans to remote applications that do not wish to go through the LMA to access these MBeans. In such cases, the JMX connector server is kept running within the application.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

**registerConfigurationMBeansOnActivation**

Controls whether the JMX management service registers configuration MBeans when it is activated.

The default is false and the MBeans are registered when a connection to the LMA is established.

Type:	Boolean
-------	---------

Required:	No
Mutability:	Restart required
Default value:	false

<b>registerRuntimeMBeansOnActivation</b>	
Controls whether the JMX management service registers runtime MBeans when it is activated.	
The default is false and the MBeans are registered when a connection to the LMA is established.	
Type:	Boolean
Required:	No
Mutability:	Restart required
Default value:	false

<b>getPIDCommand</b>	
The external command to run to retrieve the current JVM's PID.	
This command is passed to the LMA during registration so that it can expose attributes and operations for manipulating this application's process (testing whether the process is alive, killing the process for example).	
If this configuration attribute is not specified, the default behaviour is as follows:	
<ul style="list-style-type: none"> <li>• On Unix, a default command is used: <code>ps -o ppid= -p \$\$</code></li> <li>• On Windows, no default command is available, and the actual result depends on the JVM in use: <ul style="list-style-type: none"> <li>• If the JVM is a Sun 1.6+ JVM, the current PID is retrieved using an undocumented feature of the JVM (that happens to expose its PID in the name of an MXBean).</li> <li>• On other JVM types, the JVM's PID simply cannot be retrieved.</li> </ul> </li> </ul>	
When this attribute is used to specify a command, it is required that this command behaves as follows:	
<ul style="list-style-type: none"> <li>• The command's exit value should be zero.</li> <li>• The command's standard output should contain only one line which is considered to be the PID.</li> <li>• The command's error output should be empty.</li> </ul>	

If the given command does not comply with any of the above rules, the first attempt to retrieve the PID is considered to have failed, and a second attempt is made as if this command had not been specified, as described above.

Failure to retrieve the JVM's PID and to pass it on to the LMA is not necessarily a problem, except if the application runs in HA mode, in which case this service will prevent the application from starting.

Type:	String
Required:	No
Mutability:	Reactivation required

### **notificationDelayAfterRegistration**

Delay (in milliseconds) between MBean registration and sending first notification for this MBean.

This delay ensures that the notification, that is sent for MBean immediately after it is registered, is not lost. Value of this attribute should be decreased, when timeouts are encountered during configuration reload.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	50
Minimum value:	1
Maximum value:	500

## **JdbcAdapter**

### **id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes

Mutability:	Restart required
<b>class</b>	
Java class of the JDBC adapter runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.jdbc.JdbcAdapter
<b>msgTypes</b>	
The FIX message types to process with this listener.	
If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required
<b>peerApplicationManagerService</b>	
The ID of a PeerApplicationManagerService.	
This needs to be specified when the configuration contains multiple sessions that read from the same database table. The PeerApplicationManagerService handles the dispatching of FIX messages to the appropriate FIX session based on the FIX addressing fields contained in the message.	
Type:	Reference
Required:	No
Mutability:	Restart required
<b>autoForwarding</b>	
Controls whether FIX messages are forwarded to processors in the message processor chain.	

When false the adapter does not forward FIX messages to chained processors (if any). When true messages are forwarded in the normal way.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

### driver

The Java class name of the JDBC driver.

This is defined by the provider of the JDBC classes for the database being employed.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	net.sourceforge.jtds.jdbc.Driver

### url

The location of the database server and database name.

This is JDBC driver dependent.

Type:	String
Required:	No
Mutability:	Reactivation required

### user

The database username.

Type:	String
-------	--------

Required:	No
Mutability:	Reactivation required
<b>password</b>	
The database password.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>passwordStore</b>	
The ID of a PasswordStore service.	
If the "password" attribute is not given then the password will be retrived from the password store using "username". The PasswordStore service must implement the <a href="#">IPasswordStore</a> interface.	
Type:	Reference
Required:	No
Mutability:	Reactivation required
<b>peerWriter</b>	
Class that maintains a connection to the database and inserts messages received from FIX into the database.	
This class uses the configured "fixToDbInsertStatement". It must implement <a href="#">IPeerApplicationWriter</a> .	
The "peerWriter" and "peerReader" maintain separate database connections.	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	com.camerontec.catalys.server.adapter.jdbc.DefaultJdbcApplicationWriter

**peerReader**

Class that maintains a connection to the database and reads message rows from the database to be sent over FIX.

This class uses the "dbToFixPollStatement", as well as the "dbToFixUpdateStatement" to update the status column of the message. It must implement [IPeerApplicationReader](#).

The "peerWriter" and "peerReader" maintain separate database connections.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	com.camerontec.catalys.server.adapter.jdbc.- DefaultJdbcApplicationReader

**peerRecoveryInterval**

The polling interval (in milliseconds) to recover the connection to the database, should it be lost.

The JDBC adapter will try to recover the connection after this time if the connection to the database is lost, or some other SQL level error occurs which causes the connection to be reset.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	10000

**peerPollingInterval**

The period (in milliseconds) upon which FIX data is read from the database.

If set to 0 no polling is done. In this case no messages are sent to FIX.

Type:	String
Required:	No
Mutability:	Reactivation required



Default value:	0
<b>checkBeforeSend</b>	
Controls whether the JDBC adapter waits until the FIX session is logged in before sending messages to FIX.	
If true, messages are left in the database and are sent when a logged on FIX session is available.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false
<b>queryTimeoutInSeconds</b>	
The number of seconds the JDBC adapter will wait for a query to execute.	
If the limit is exceeded, an Exception is thrown. Zero means there is no limit. If not set the timeout defaults to the underlying database timeout.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>fixToDbTags</b>	
A comma-separated list of FIX tags to parse from FIX messages to store in the database.	
No data type checking is performed.	
Type:	String
Required:	No
Mutability:	Restart required

**fixToDbFile**

A file of FIX tags in the format specified by "fixToDbFileFormat" to parse from FIX messages to store in the database.

Type:	String
Required:	No
Mutability:	Restart required

**fixToDbFileFormat**

Format of the "fixToDbFile" file.

Can be any combination of desc (human readable description), tag (tag value), type (the data type of the corresponding field in the database) and/or req (whether NULL is permitted or not).

Type:	String
Required:	No
Mutability:	Restart required
Default value:	desc,tag,type,req

**dbToFixTags**

A comma separated list of tags corresponding to the columns of the database table being polled for messages to send to FIX.

A reserved tag 'ignored' may be used to instruct the JdbcAdapter to ignore the tag and not attempt to insert it into the FIX message. In the templates the house-keeping columns 'msgId' and 'status' are configured using this reserved tag. The tags specified must constitute a valid FIX message, so at least the message type tag (35) must be present.

Type:	String
Required:	No
Mutability:	Restart required

**dataTypesFile**

A file which provides a data type mapping between the specific database and JDBC.

Only required if data type information is provided in the "fixToDbFile" file. This file should also be placed in the application classpath.

If no such information exists in the "fixToDbFile" file no datatype conversion is attempted and all values are VARCHARs.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	sybaseToJdbc.properties

#### **fixToDbInsertStatement**

An SQL statement or the name of a stored procedure to insert messages into the database.

If it is the name of a stored procedure, then the "fixToDbInsertWithStoredProc" flag must be set to true.

No JDBC placeholder parameters ('?') are necessary in either context. The appropriate number of placeholder parameters is automatically inserted when the statement is created based on the number of FIX tags returned upon message transformation.

By default this statement is executed as an update and is expected to return a row count of 1.

Type:	String
Required:	No
Mutability:	Reactivation required

#### **fixToDbInsertWithStoredProc**

Indicates that "fixToDbInsertStatement" is a stored procedure name.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

**dbToFixPollStatement**

An SQL statement or the name of a stored procedure to read message data from the database.

If it is the name of a stored procedure, then the "dbToFixPollWithStoredProc" flag must be set to true.

Execution of this statement is expected to return zero or more rows which each row containing FIX message data. Each row is expected to contain the house-keeping columns 'msgld' and 'status'. The 'msgld' column must be unique.

The writing application should set the 'status' column to '0' initially.

Each row of message data transformed according to "dbToFixTags" and then sent to FIX.

Type:	String
Required:	No
Mutability:	Reactivation required

**dbToFixPollWithStoredProc**

Indicates that "dbToFixPollStatement" is a stored procedure name.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

**dbToFixUpdateStatement**

An SQL statement or stored procedure name to update the 'status' of a row identified by 'msgld'.

If it is the name of a stored procedure, then the "dbToFixUpdateWithStoredProc" flag must be set to true.

Its execution is expected to return a row count of 1. Correspondingly, two place holder parameters must be explicitly included for this template in the order ('status', 'msgld'). Possible values of the 'status' column are:

- 0 - New (row has not yet been read)

- 1 - Read (row has been read and transformed but not sent)
- 2 - Sent (row has been read, transformed and sent)
- 3 - Error (row contains error, or could not be sent for some other reason). This is a permanent error.

Type:	String
Required:	No
Mutability:	Reactivation required

#### dbToFixUpdateWithStoredProc

Indicates that "dbToFixUpdateStatement" is a stored procedure name.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

#### dbToFixUseStrictSequencing

Controls whether the adapter sends messages to FIX in a strict sequence determined by the 'msgId' column.

When true, if the last row sent had a 'msgId' of 10 then the adapter will not send any more rows until a row with 'msgId' 11 is inserted.

When false, messages are sent as they are inserted into the database regardless of their 'msgId' column (although 'msgId' ordering is still preserved when multiple rows are added between polls).

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

**dbToFixMinMsgId**

Specifies the minimum 'msgId' to use when calculating the next expected 'msgId', used in strict sequencing mode.

This value is only used when it is greater than or equal to the result of the "dbToFixInitStatement".

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	1

**dbToFixInitStatement**

An SQL statement or stored procedure name which returns the maximum 'msgId' of all rows which have either been marked as Sent or Error, used in strict sequencing mode.

This statement is executed at initialisation to determine the 'msgId' to expect next. The result of this statement should be a single row containing a single column being the maximum 'msgId'. The JDBC adapter assumes that the next 'msgId' is this value plus 1.

Type:	String
Required:	No
Mutability:	Reactivation required

**dbToFixInitWithStoredProc**

Indicates that "dbToFixInitStatement" is a stored procedure name.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

**transformer**

Class used to transform database objects to and from FIX messages.

This class must implement [IMessageTransformer](#).

The transformer is expected to accept a ResultSet for the outbound direction and return a FieldValueInfo list for the inbound direction.

Since this adapter sets the "storeMessages" attribute to false to enable message reuse, transformers should not store references to the FIX message object.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	com.camerontec.catalys.server.adapter.jdbc.FlexibleTagTransformer

## JmsConsumer

**id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

**class**

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value

Default value:	com.camerontec.catalys.server.adapter.jms.JmsConsumer
<b>type</b>	
<p>The JMS destination type.</p> <p>For point-to-point configurations, this should be set to 'queue', and for publish-and-subscribe communications, this should be set to 'topic'.</p>	
Type:	String
Required:	Yes
Mutability:	Restart required
Legal values:	topic   queue
<b>durableName</b>	
<p>Name of a subscriber to which to send messages, so that they are available should this JmsConsumer go down.</p> <p>Messages will be retrieved from this subscriber once the JmsConsumer recovers.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>msgSelector</b>	
<p>SQL style select logic that JMS applies to message properties.</p> <p>For example, NewsType='Sports' OR NewsType='Opinion'.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>connectionId</b>	



Identify this JmsConsumer if the JMS servers are installed on different machines, but have an identical destination string.

Type:	String
Required:	No
Mutability:	Restart required

### destination

The name of the destination to receive from.

Typically, destinations are set up using an administration tool provided by your JMS provider.

Type:	String
Required:	Yes
Mutability:	Restart required

### initialWait

Number of ms to wait between creating a JMS session and listening for messages.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	5000

### factory

The name of the JMS connection factory used to access JMS destinations and create JMS messages.

Typically, this is set up using an administration tool provided by your JMS provider.

Type:	String
Required:	No
Mutability:	Restart required

**factory.parameters**

Parameters to supply to the JMS connection factory constructor.

The value of this attribute is a "factory.parameters.separator" separated list of pairs of fully qualified java type and value, separated by single space. For example: "java.lang.String foo,java.lang.Integer 3" passes "String foo" and "Integer 3" to the factory (when factory.parameters.separator is equal to a comma).

Only required by some factories, and if JNDI is not being used to load the JMS connection factory.

Type:	String
Required:	No
Mutability:	Restart required

**factory.parameters.separator**

Separator used to delimit individual parameters from each another in the "factory.parameters" attribute.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	,

**use.jndi**

Controls whether JNDI is used to load the JMS connection factory.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	true
Legal values:	true   false

**reconnectDelay**

The number of milliseconds to wait between attempts to reconnect to JMS after a JMS Exception.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	10000

### transformer

The transformer used to transform JMS messages objects into FIX messages.

This class must implement [IMessageTransformer](#).

Type:	String
Required:	No
Mutability:	Restart required
Default value:	com.camerontec.catalys.server.adapter.jms.DefaultJMSTransformer

### username

Username that may be required to connect to JMS.

Type:	String
Required:	No
Mutability:	Restart required

### password

Password that may be required to connect to JMS

Type:	String
Required:	No
Mutability:	Restart required

**passwordStoreServiceName**

The name of a PasswordStore service.

If the "password" attribute is not given then the password will be retrieved from the password store using "username". The PasswordStore service must implement the [IPasswordStore](#) interface.

Type:	Reference
Required:	No
Mutability:	Reactivation required

**replyOnError**

If true and an outbound FIX message cannot be delivered, a response error message is sent over the replyTo queue value from the original message. Default value is false.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	false

## JmsProducer

**id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

**class**

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.jms.JmsProducer
<b>msgTypes</b>	
The FIX message types to process with this adapter. If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required
<b>type</b>	
The JMS destination type.	
For point-to-point configurations, this should be set to 'queue'.	
Type:	String
Required:	Yes
Mutability:	Restart required
Legal values:	topic   queue
<b>destination</b>	
The name of the destination to receive from.	
Typically, destinations are set up using an administration tool provided by your JMS provider.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>factory</b>	

The name of the JMS connection factory used to access JMS destinations and create JMS messages.

Typically, this is set up using an administration tool provided by your JMS provider.

Type:	String
Required:	No
Mutability:	Restart required

### **factory.parameters**

Parameters to supply to the JMS connection factory constructor.

The value of this attribute is a "factory.parameters.separator" separated list of pairs of fully qualified java type and value, separated by single space. For example: "java.lang.String foo,java.lang.Integer 3" passes "String foo" and "Integer 3" to the factory (when factory.parameters.separator is equal to a comma).

Only required by some factories, and if JNDI is not being used to load the JMS connection factory.

Type:	String
Required:	No
Mutability:	Restart required

### **factory.parameters.separator**

Separator used to delimit individual parameters from each another in the "factory.parameters" attribute.

Type:	String
Required:	No
Mutability:	Restart required

### **use.jndi**

Controls whether JNDI is used to load the JMS connection factory.

Type:	String
Required:	No

Mutability:	Restart required
Default value:	true
Legal values:	true   false
<b>reconnectDelay</b>	
The number of milliseconds to wait between attempts to reconnect to JMS after a JMS Exception.	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	10000
<b>transformer</b>	
The transformer used to transform FIX messages into JMS message objects.	
This class must implement <a href="#">IMessageTransformer</a> .	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	com.camerontec.catalys.server.adapter.jms.DefaultJMSTransformer
<b>username</b>	
Username that may be required to connect to JMS	
Type:	String
Required:	No
Mutability:	Restart required
<b>password</b>	
Password that may be required to connect to JMS	

Type:	String
Required:	No
Mutability:	Restart required
<b>passwordStoreServiceName</b>	
<p>The name of a PasswordStore service.</p> <p>If the "password" attribute is not given then the password will be retrieved from the password store using "username". The PasswordStore service must implement the <a href="#">IPasswordStore</a> interface.</p>	
Type:	Reference
Required:	No
Mutability:	Reactivation required
<b>timeToLive</b>	
<p>The MessageProducer timeToLive.</p> <p>The default length of time in milliseconds from its dispatch time that a produced message should be retained by the message system.</p>	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	0

## JournalingPersister

<b>id</b>	
<p>ID of configuration element</p> <p>Must be unique across the same configuration element types.</p>	
Type:	String



Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	com.camerontec.catalys.core.session.persistence.JournalingPersister
<b>baseDirectoryName</b>	
<p>The name of the directory where persistence files are stored.</p> <p>This is used as the base path name for the directories associated with this persister. Persistence directories are created with names constructed from this base name plus suffixes corresponding to the element type, as follows:</p> <ul style="list-style-type: none"> <li>• Output messages: base name plus -outMsg</li> <li>• Input sequence number: base name plus -inSeqNum</li> <li>• Attributes: base name plus -atts</li> <li>• Input messages: base name plus -inMsg (this file is only present if 'persistInputMessages' has been specified)</li> </ul>	
Type:	String
Required:	Yes
Mutability:	Reactivation required
<b>baseCollectionName</b>	
<p>The base name used for collections associated with this persister, used when registering the collections and exchanged between cluster nodes for the purpose of replication.</p>	

If it is not specified, the value of 'baseDirectoryName' is used instead. If this node is part of a High Availability cluster then better performance is achieved by specifying a succinct value for this attribute, because the value of 'baseDirectoryName' can be long (especially when it specifies an absolute path).

Names must be unique across all collections bound to the Collections Registry.

Collection names are constructed from this base name plus suffixes corresponding to the element type:

- Output messages: base name plus -outMsg
- Input sequence number: base name plus -inSeqNum
- Attributes: base name plus -atts
- Input messages: base name plus -inMsg (this collection is only created if 'persistInputMessages' has been specified)

Type:	String
Required:	No
Mutability:	Reactivation required

### **persistInputMessages**

Controls whether input messages are persisted.

Note that it is not necessary to persist input FIX messages in order to maintain state and doing so is likely to incur a performance penalty.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### **persistOutputMessages**

Controls whether output messages are persisted.

Defaults to true.

Type:	Boolean
-------	---------

Required:	No
Mutability:	Reactivation required
Default value:	true

**outSelectorRefID**

Reference to the ID of a declared Selector.

This allows control of which outbound messages are persisted.

Messages that are not selected will not be persisted and therefore will not be retransmitted if requested by the counter-party. Upon receipt of a ResendRequest message, a FIX SequenceReset-GapFill message will be sent instead.

For example, when sending execution reports as well as market data on a single connection, you may elect to not persist market data messages so that stale data are not sent in response to a ResendRequest message.

Type:	Reference
Required:	No
Mutability:	Reactivation required

**inSelectorRefID**

Reference to the ID of a declared Selector.

This allows control of which inbound messages are persisted (when 'persistInputMessages' is enabled). Messages that are not selected will not be persisted. Defaults to all messages being selected.

Type:	Reference
Required:	No
Mutability:	Reactivation required

## Link

**id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### **class**

Java class of this element's Runtime object.

Type:	String
Required:	Yes
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.routing.LinkBase

### **useDeliverToRouting**

Determines whether deliverTo routing or targetID routing is used.

If false, the DeliverTo party fields are ignored.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	true

### **concurrentRouting**

Determines whether multiple sessions can route messages in different threads.

If true, multiple sessions are allowed to route messages concurrently, potentially to the same target session. This allows for more efficient processing but requires care with the implementation of message processors in the processing chain of the target sessions due to potential thread-safety issues.

If false (the default), only one session may route messages to another session at a time. This is easier to implement, but it is possible for a sluggish target session to slow down the routing of other sessions.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### **anyToAny**

Allows any session to be routed to any other.

If true, there must be only one empty Link element, and sessions must not specify side attribute.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### **directed**

Determines whether routing is in one direction only.

If true, routing is in one direction only - from connection points in the first group to points in the second group. Messages are not routed back.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### **useOnBehalfOfSendingTime**

Determines whether the OnBehalfOfSendingTime field is added to each routed message (when useDeliverToRouting is true).

If true and useDeliverToRouting is true, OnBehalfOfSendingTime (FIX tag 370) contains the value of the SendingTime of the message before it was routed.

Note: this FIX field is only applicable to FIX versions 4.2 and 4.3.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### useHopFields

Determines whether the Hops fields (NoHops, HopCompID, HopRefID and HopSendingTime) are added to each routed message.

If true, NoHops (FIX tag 627), HopCompID (FIX tag 628), HopSendingTime(FIX tag 629), and HopRefID (FIX tag 630) are added to each routed message.

Note: these FIX fields are only applicable for FIX versions later than FIX 4.3.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### usePartialConnectionPointMatching

Determines whether all fields of the party must be matched for a

If false, If false, FIX messages are routed only if all of the standard FIX message address tags (Target/ Sender CompID/SubID/LocationID) contained in the message match exactly the configured address tags of the routing target session.

If true, the address tags contained in the FIX message need only be descendants of the configured address tags of the routing target session. That is, if the FIX message contains address tags in addition to those defined by the routing target session the message will still be routed. Care must be taken with this option since it opens the possibility of ambiguous routing target sessions.

Type:	Boolean
-------	---------

Required:	No
Mutability:	Reactivation required
Default value:	false
<b>directoryName</b>	
<p>The base name of the directory in which session routing information is persisted (used with Rules Engine Routing, when <code>autoRoutingResponse</code> is true).</p> <p>Information is persisted per destination session. This directory name is extended by the session ID. Refer to <a href="#">ActionRouteTo</a>.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>orderStatusesToArchive</b>	
<p>A list of values of order status that indicate that an order is closed and can be archived.</p> <p>Used with Rules Engine Routing, when <code>autoRoutingResponse</code> is true. Refer to <a href="#">ActionRouteTo</a>.</p> <p>If no statuses are specified then the following default list is used: Canceled, Expired, Done for Day, Filled, Rejected. Value NONE means no archiving.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>endOfDayClearingCondition</b>	
<p>A Rules Expression specifying a condition which determines which orders to purge at end of day.</p> <p>Used with Rules Engine Routing, when <code>autoRoutingResponse</code> is true. Refer to <a href="#">ActionRouteTo</a>.</p> <p>This parameter allows specification of a condition or set of conditions that determine whether a message will be cleared from the Market Mirror storage at the end of the day. This specification can be any expression allowed by the <a href="#">Rules Expression Language</a>.</p>	

The following value clears orders which were only valid for the day (that is they do not have a `TimeInForce` field, or its value is set to zero): `"!isFieldPresent(TimeInForce) || (isFieldPresent(TimeInForce) && TimeInForce == 0)"`

This example clears orders that have closed (that is, the value of their `OrdStatus` field is one of: Canceled, Filled, Expired, Done for Day, or Rejected): `"matches(OrdStatus, '[42C38])"`

These expressions can also be combined logically.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	<code>matches(OrdStatus, '[42C38])    !isFieldPresent(TimeInForce)    TimeInForce == 0</code>

## ListenerMessageBuffer

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	<code>com.camerontec.catalys.server.storeandforward.ListenerMessageBuffer</code>



**msgTypes**

The FIX message types to process with this listener. If not set, all message types will be processed.

Type:	String
Required:	No
Mutability:	Restart required

**directoryName**

The name of the directory that is used to hold messages that have been held by the buffer.

The name can be an absolute path or a relative one. The directory is automatically cleared when the buffer becomes empty so there is no maintenance required.

Type:	String
Required:	Yes
Mutability:	Restart required

**clearOnReset**

Indicates whether buffers should clear out all messages when the sequence numbers of its associated session are reset.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

**clearOnEOD**

Indicates whether buffers should clear out all messages when the end of day procedure is performed on the session.

Type:	String
-------	--------

Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

**alwaysBuffer**		
Indicates that the message buffer should always store all messages and deliver them asynchronously, regardless of whether synchronous delivery would have been successful or not.  If false then buffering will only be triggered if message delivery fails. Only then will the message buffer processor insert the failed messages in the buffer. Another independent thread, the retry thread, polls the buffer every retryInterval and tries to deliver the messages asynchronously. Once the retry thread delivers all messages then messages are not buffered anymore.  If true then the message buffer processor will always place all messages in the buffer. Another independent thread, the delivery thread, delivers the messages as soon as they are placed in the buffer. When the buffer is empty the delivery thread blocks waiting for new message(s) to be inserted in the buffer. If the delivery thread cannot deliver the message(s) then it sleeps an amount of time determined by retryInterval.		
Type:	String	
Required:	No	
Mutability:	Restart required	
Default value:	false	
Legal values:	true	false
**rejectIfNotSelected**		
Indicates what to do with received messages which are not selected while buffering.  If true, unselected messages received while buffering cause an exception to be thrown (i.e. they are rejected). If false, they are ignored.		
Type:	String	
Required:	No	

Mutability:	Reactivation required
Default value:	true
Legal values:	true   false
<b>reportSizeAttribute</b>	
<p>Name of attribute used to report changes to the number of messages held in the buffer.</p> <p>If set, an attribute of that name will be used to store the number of messages in the buffer. Whenever the value of that attribute changes, an <code>AttributeChangeEvent</code> is fired which other objects can listen to by calling <code>addAttributeChangeListener</code>.</p> <p>If not set then changes to the buffer size are not reported.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>retryInterval</b>	
<p>The number of milliseconds between send retry attempts.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	10000
<b>selectorRefID</b>	
<p>The ID of a configured selector, which controls which messages are buffered.</p> <p>Messages received while buffering which are not selected are rejected by default (see the <code>rejectIfNotSelected</code> attribute).</p> <p>If not set then all messages are buffered.</p>	
Type:	Reference

Required:	No
Mutability:	Restart required
<b>compactWhenEmpty</b>	
Indicates whether buffer's persistent collection should be compacted whenever the buffer's size reaches 0.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

## ListenerMessageThrottle

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value

Default value:	com.camerontec.catalys.server.storeandforward.ListenerMessageThrottle
<b>msgTypes</b>	
The FIX message types to process with this listener. If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required
<b>directoryName</b>	
The name of the directory that is used to hold messages that have been held by the throttle.	
The name can be an absolute path or a relative one. The directory is automatically cleared when the throttle becomes empty so there is no maintenance required.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>selectorRefID</b>	
The ID of a configured selector, which controls which messages are buffered.	
Messages received while throttling which are not selected are rejected by default (see the <code>rejectIfNotSelected</code> attribute).	
If not set then all messages are buffered.	
Type:	Reference
Required:	No
Mutability:	Restart required
<b>rejectIfNotSelected</b>	
Indicates what to do with received messages which are not selected while buffering.	

If true, unselected messages received while buffering cause an exception to be thrown (i.e. they are rejected). If false, they are ignored.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

### interval

The interval over which to count the number of allowable messages.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	1

### intervalTimeUnit

The time unit for the interval attribute. Allowed values are either 's' for seconds or 'ms' for milliseconds.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	s
Legal values:	s   ms

### messagesPerInterval

The number of messages to allow in the given interval.

Type:	String
-------	--------

Required:	No
Mutability:	Reactivation required
Default value:	10
<b>compactWhenEmpty</b>	
Indicates whether throttle's persistent collection should be compacted whenever the throttle's size reaches 0.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

## MSMQMessageListener

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No

Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.msmq.MSMQProducer
<b>msgTypes</b>	
The FIX message types to process with this listener. If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required
<b>queue.formatName</b>	
The format name of the MSMQ queue that this adapter writes to.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>queue.isTransactional</b>	
Whether or not the queue is transactional. Defaults to false.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false
<b>reconnectDelay</b>	
Number of milliseconds to wait before reconnecting after a disconnection.	
Type:	String



Required:	No
Mutability:	Reactivation required
Default value:	10000
<b>transformer</b>	
Class used to transform FIX messages into MSMQ messages objects. This class must implement <a href="#">IMessageTransformer</a> .	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.core.message.StandardFixTransformer
<b>autoForwarding</b>	
Controls whether messages are automatically forwarded to other listeners in the chain. If true, messages are automatically forwarded.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

## MSMQMessageSource

<b>id</b>
ID of configuration element Must be unique across the same configuration element types.

## Configuration Reference

Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.msmq.MSMQConsumer
<b>queue.formatName</b>	
The format name of the MSMQ queue that this adapter reads from.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>queue.isTransactional</b>	
Whether or not the queue is transactional. Defaults to false.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false
<b>reconnectDelay</b>	
Number of milliseconds to wait before reconnecting after a disconnection.	

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	10000
<b>transformer</b>	
Class used to transform MSMQ messages objects into FIX messages.	
This class must implement <a href="#">IMessageTransformer</a> .	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.core.message.StandardFixTransformer

## MarketMirrorPopulator

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No

Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.marketstate.MarketMirrorPopulator
<b>msgTypes</b>	
The FIX message types to process with this listener. If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required
<b>marketMirrorId</b>	
The ID of an existing market mirror service to populate.	
If not specified, a new market mirror service is created, the ID of which is formed from the class name of the market mirror service and the ID of this market mirror populator.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>marketMirrorClass</b>	
The class to use for the market mirror service (if not populating an existing market mirror service).	
This class must implement the <a href="#">IMarketMirror</a> interface. Catalys provides <code>FixMarketMirror</code> and <code>FixMarketMirrorWithOrderArchive</code> . If no <code>marketMirrorClass</code> and no <code>marketMirrorId</code> is given then <a href="#">FixMarketMirror</a> is used.	
If <code>storesMessages</code> attribute is set to false to enable message reuse, <code>marketMirrorClass</code> may not store any references to the FIX message object.	
Type:	String
Required:	No
Mutability:	Reactivation required

**orderStatusesToArchive**

A list of values of order status that indicate that an order is closed and can be archived.

Only used if the market mirror populator is capable of archiving. This is the case when the marketMirrorClass attribute has been set to [FixMarketMirrorWithOrderArchive](#) or a class that implements [IMarketMirrorWithOrderArchive](#).

If no statuses are specified then the following default list is used: Canceled, Expired, Done for Day, Filled, Rejected.

Type:	String
Required:	No
Mutability:	Reactivation required

**archiveFileName**

Name of archive log.

Only used if the market mirror populator is capable of archiving. This is the case when the marketMirrorClass attribute has been set to [FixMarketMirrorWithOrderArchive](#) or a class that implements [IMarketMirrorWithOrderArchive](#).

Type:	String
Required:	No
Mutability:	Reactivation required

**archiveImplClass**

Market mirror order archive class name.

Only used if the market mirror populator is capable of archiving. This is the case when the marketMirrorClass attribute has been set to [FixMarketMirrorWithOrderArchive](#) or a class that implements [IMarketMirrorWithOrderArchive](#).

This class must implement the [IOrderArchive](#) interface.

If not specified [OrderArchiveLog](#) will be used.

If storesMessages attribute is set to false to enable message reuse, archiveImplClass may not store any references to the FIX message object.

Type:	String
Required:	No
Mutability:	Reactivation required
<b>directoryName</b>	
<p>Name of directory in which to persist FIX session traffic, so that it is not lost when the Catalys Node restarts.</p> <p>This directory will be replicated across the cluster if the HA option is active. Upon restart the <code>MarketMirrorPopulator</code> will re-populate the market mirror from the contents of this file.</p> <p>The files in this directory are compacted as part of the application's end of day procedure, which can be <a href="#">scheduled</a> or called via the <a href="#">CLI</a>.</p> <p>To configure the persistence files, the <code>MarketMirrorPopulator</code> can use a nested Properties element, the contents of which can contain any of the <a href="#">List Collection Properties</a>.</p> <p>Users can also plug in their own persistence solution by installing a custom collection registry service. All <code>MarketMirrorPopulator</code> attributes are made available to the custom collection registry in order to control customer collection creation semantics inside the custom collection registry.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>endOfDayClearingCondition</b>	
<p>A Rules Expression specifying a condition which determines which orders to purge from the market mirror storage at end of day.</p> <p>This parameter allows specification of a condition or set of conditions that determine whether a message will be cleared from the Market Mirror storage at the end of the day. This specification can be any expression allowed by the <a href="#">Rules Expression Language</a>.</p> <p>The following value clears orders which were only valid for the day (that is they do not have a <code>TimeInForce</code> field, or its value is set to zero): <code>"!isFieldPresent(TimeInForce)    (isFieldPresent(TimeInForce) &amp;&amp; TimeInForce == 0)"</code></p> <p>This example clears orders that have closed (that is, the value of their <code>OrdStatus</code> field is one of: Canceled, Filled, Expired, Done for Day, or Rejected): <code>"matches(OrdStatus, '[42C38])"</code></p>	

These expressions can also be combined logically.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	matches(OrdStatus, '[42C38]')    !isFieldPresent(TimelnForce)    TimelnForce == 0

### storesMessages

Controls whether references to messages are stored by this processor.

If false, then messages objects can be reused. In case when Market Mirror or Archive Implementation class stores references to the message this attribute should be set to true. The default is false.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

## MemoryPersister

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### class

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.core.session.persistence.MemoryPersister

## MessageProcessingInstance

### refid

ID of message processing template to instantiate.

Type:	Reference
Required:	Yes
Mutability:	Restart required

## MessageProcessingTemplate

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

## MessageProcessorReference

### id

ID of configuration element



Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### refid

The ID of message processor configuration this element refers to.

Type:	Reference
Required:	Yes
Mutability:	Reactivation required

## MessageToStringFormat

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### class

Java class of this element's Runtime object.

Type:	String
Required:	Yes
Mutability:	Reactivation required

### format

The toString format to use.

Type:	String
Required:	Yes
Mutability:	Reactivation required
Legal values:	SUMMARY   SUMMARY_VERBOSE   FULL   VERBOSE   RAW_INPUT

## MsgTypeSelector

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### class

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.messageselector.MsgTypeSelector

### includeMsgTypes

A list of message types to include (e.g., 'D 8 G').

Type:	String
Required:	No

Mutability:	Reactivation required
<b>excludeMsgTypes</b>	
A list of message types to exclude (e.g., 'D 8 G').	
Type:	String
Required:	No
Mutability:	Reactivation required

## MultiHopRoutingRules

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.test.MultiHopRoutingRules
<b>msgTypes</b>	
The FIX message types to process. If not set, all message types are processed.	
Type:	String

Required:	No
Mutability:	Restart required
<b>hideRoutingFields</b>	
Controls whether routing fields are removed from outbound messages after they have been routed.	
If true, routing fields are removed. Routing fields include:	
<ul style="list-style-type: none"> <li>• DeliverToCompID (FIX tag 128), DeliverToSubID (FIX tag 129), DeliverToLocationID (FIX tag 145)</li> <li>• OnBehalfOfCompID (FIX tag 115), OnBehalfOfSubID (FIX tag 116), OnBehalfOfLocationID (FIX tag 144)</li> <li>• NoHops (FIX tag 627), HopCompID (FIX tag 628), HopSendingTime (FIX tag 629), HopRefID (FIX tag 630)</li> </ul>	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

## NullMessageProcessor

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.processor.NullMessageProcessor
<b>msgTypes</b>	
The FIX message types to process. If not set, all message types are processed.	
Type:	String
Required:	No
Mutability:	Restart required

## OrderFieldAccessor

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.test.OrderFieldAccessor

**msgTypes**

The FIX message types to process. If not set, all message types are processed.

Type:	String
Required:	No
Mutability:	Restart required

## OuchMessageProcessor

**id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

**class**

Java class of the Runtime object corresponding to this configuration object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.ouch.OuchMessageProcessor

**msgTypes**

The FIX message types to process with this processor. If not set, all message types will be processed.

Type:	String
Required:	No

Mutability:	Restart required
<b>ouchSessionManagerServiceRefid</b>	
OUCH Session Service reference ID	
Type:	Reference
Required:	Yes
Mutability:	Reactivation required
<b>referenceDataServiceRefid</b>	
reference Data Service reference ID	
Type:	Reference
Required:	Yes
Mutability:	Reactivation required
<b>customerInfo</b>	
A unique value for this FIX session to insert into the OUCH customer info field.	
Type:	String
Required:	Yes
Mutability:	Reactivation required
<b>removeClosedOrders</b>	
Controls whether closed orders are removed from cache.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true

**ouchInboundMessageFactoryClass**

OUCH inbound(in respect to OUCH server) message factory class. Should implement OuchInboundMessageFactory interface.

Type:	String
Required:	Yes
Mutability:	Reactivation required

**orderIdFormatterClass**

OUCH exchange order id to FIX order id formatter.

Type:	String
Required:	No
Mutability:	Reactivation required

## OuchSessionManagerService

**id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

**class**

Java class of the Runtime object corresponding to this configuration object.

Type:	String
Required:	No



Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.ouch.- OuchSessionManagerService
<b>orderTokenGeneratorClass</b>	
Order token generator class	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.server.adapter.asx.ouch.orderTokenGenerator.- PositiveNumberOrderTokenGenerator
<b>soupTCPSessionServiceRefids</b>	
soupTCP Session Service reference IDs comma separated if more then one configured	
Type:	String
Required:	Yes
Mutability:	Reactivation required
<b>ouchOutboundMesageFactoryClass</b>	
OUCH outbound(in respect to OUCH server) message factory class. Should implement OuchOutboundMessageFactory interface.	
Type:	String
Required:	Yes
Mutability:	Reactivation required

## PGP\_DES\_MD5

**class**

Java class of this element's Runtime object.

Type:	String
Required:	Yes
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.core.crypto.CryptixCrypto

### **encryptmethod**

Encryption method used by this element.

For this PGP\_DES\_MD5 element, this attribute has the fixed value of '5'.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	5

### **publickeyring**

The path and name of the file containing the public encryption key.

Type:	String
Required:	Yes
Mutability:	Reactivation required

### **secretkeyring**

The path and name of the file containing the private (secret) encryption key.

Type:	String
Required:	Yes
Mutability:	Reactivation required

**passphrase**

The passphrase associated with your private (secret) encryption key.

Type:	String
Required:	Yes
Mutability:	Reactivation required

**myprivatekeyname**

The name associated with your private (secret) encryption key. Required.

Type:	String
Required:	Yes
Mutability:	Reactivation required

**otherspublickeyname**

The name associated with your counterpart's public encryption key. Required.

Type:	String
Required:	Yes
Mutability:	Reactivation required

## PeerApplicationManagerService

**id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

**class**

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.PeerApplicationManagerService

**waitForApplicationStartup**

Controls whether message reader waits until application is fully started before reading messages.

If set to false(default), there is a chance of message loss during HA fail over. Message reader may start reading messages before all sessions are activated. Setting this value to true is recommended.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

## PerformanceMessageTester

**id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

**class**

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.test.PerformanceMessageTester

### msgTypes

The FIX message types to process. If not set, all message types will be processed.

Type:	String
Required:	No
Mutability:	Restart required

### storesMessages

Controls whether references to messages are stored by this processor.

If false, then messages objects can be reused, and the processor will operate more efficiently.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

### cycles

The number of messages used to calculate inbound and outbound messages rates.

Type:	String
Required:	No

Mutability:

Reactivation required

## Ping

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:

String

Required:

Yes

Mutability:

Restart required

### destinations

List of destinations to ping.

Type:

Object

Required:

Yes

Mutability:

Restart required

## PrimaryValidityCheckScript

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:

String

Required:

Yes

Mutability:

Restart required

### command

The location of the script to run to validate the primary.

Type:	String
Required:	Yes
Mutability:	Restart required

## Properties

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### properties

The set of properties contained within this element.

Type:	Object
Required:	No
Mutability:	Reactivation required

## PropertiesReference

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
-------	--------

Required:	Yes
Mutability:	Restart required
<b>refid</b>	
The ID of properties configuration this element refers to.	
Type:	Reference
Required:	Yes
Mutability:	Reactivation required

## RandomFileSource

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.test.RandomFileSource
<b>checkBeforeSend</b>	
Controls whether messages are only sent when the session is logged on.	



## Configuration Reference

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

### interval

Milliseconds between sending each message

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	2000

### inputDir

Directory from which to randomly read files that contain FIX messages.

Type:	String
Required:	No
Mutability:	Reactivation required

### tagSeperator

Separator between each field in the message files.

Type:	String
Required:	No
Mutability:	Reactivation required

### readLineByLine

Controls whether files contain one message per line or one message per file.

If true, then the files contain one message per line. If false, the files contain one message per file.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

### transformer

Class used to transform messages to and from the file format into FIX message format.

This class must implement [IMessageTransformer](#).

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.core.message.StringToMessageTransformer

## RejectMapper

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### class

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.routing.RejectMapper
<b>msgTypes</b>	
The FIX message types to process with this listener. If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required
<b>directoryName</b>	
Name of the directory used to persist the mapping of sequence numbers, so that they are not lost when the node restarts.	
The persisted map can be configured by supplying the RejectMapper with a nested Properties element containing <a href="#">Map Collection Properties</a> .	
The files in this directory are compacted as part of the application's end of day procedure, which can be <a href="#">scheduled</a> or called via the <a href="#">CLI</a> .	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>forwardOnRejectMappingFailure</b>	
Whether or not to forward reject messages even if they could not be properly mapped.	
Type:	String
Required:	No
Mutability:	Reactivation required

Default value:	true
Legal values:	true   false
<b>hideUnmappedRefseqnums</b>	
Whether or not to set the RefSeqNum to '0' if a sequence number could not be mapped successfully.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

## RejectMessageListener

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.test.RejectMessageListener

**msgTypes**

The FIX message types to process. If not set, all message types are processed.

Type:	String
Required:	No
Mutability:	Restart required

## RelinquishingPrimaryScript

**id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

**command**

The location of the script to run when relinquishing primary status.

Type:	String
Required:	Yes
Mutability:	Restart required

## ReservedCore

**thread**

Thread name

Type:	String
-------	--------

Required:	Yes
Mutability:	Restart required
<b>cpu</b>	
CPU ID	
Type:	Integer
Required:	Yes
Mutability:	Restart required

## RmiAdapter

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.rmi.RmiAdapter
<b>msgTypes</b>	
The FIX message types to process with this listener. If not set, all message types will be processed.	

Type:	String
Required:	No
Mutability:	Restart required
<b>name</b>	
The RMI name associated with this RmiAdapter (and hence this instance of the Catalys Server).	
Type:	String
Required:	No
Mutability:	Restart required
<b>port</b>	
The port on which the RMI registry is listening.	
If not specified, the standard default RMI registry port number is used (1099).	
Type:	String
Required:	No
Mutability:	Restart required
<b>directoryName</b>	
The name of the directory which will be used to log incoming messages.	
This is required in order to retrieve messages using the remote getMessage methods rather than through listener objects and callbacks. If not specified, attempts to call any of the getMessage methods on a remote session will throw an exception.	
Type:	String
Required:	No
Mutability:	Restart required
<b>newRegistry</b>	
Determines whether an RMI registry is started, or an existing one is used.	

If true (the default) an RMI registry is automatically started, listening on the specified port. If false an existing RMI registry (started using the standard Java rmiregistry utility for example) is used.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	true
Legal values:	true   false

### **rmiClientTimeout**

Timeout, in milliseconds, applied to all reads on the underlying client TCP socket.

If not specified then no timeout is applied, as per the default implementation.

All remote RMI calls are marshalled across an underlying TCP socket which by default does not have a read timeout set. This means that the caller can potentially block forever in some failure modes. If specified, this timeout is applied to all Remote methods exposed by the RMI adapter.

This timeout should not be set too low as the underlying TCP can recover from transient errors. If set too low then clients are potentially forced to handle false timeouts.

When a timeout occurs the call in progress is interrupted and a `RemoteException` is thrown with the underlying cause set to `SocketTimeoutException`. At this point the client should assume there is a problem and should attempt error recovery. By default, the RMI adapter removes clients from its listener list when an error occurs when attempting a call on that listener. The client has no way of knowing whether this has occurred. Therefore, the client should initiate recovery by calling `addFixListener` again until it is successful. Note that the RMI adapter will ignore attempts to add duplicate listeners, therefore this call is safe at any time.

Client applications, which are typically FIX listeners, may use a similar technique if a timeout is required on remote calls made from the adapter to the client. Unless the client organises for such a timeout to be applied, then all remote calls from the adapter to the client have no timeout and may potentially block forever.

See the Java API documentation on the `RMISocketFactory` class for more details.

Type:	String
Required:	No



Mutability:	Restart required
-------------	------------------

<b>rmiFixedPort</b>	
<p>Specifies that this fixed port value should be used whenever RMI requires a new listening port.</p> <p>Is only needed when the RMI adapter is on the other side of a firewall from the client.</p> <p>By default, RMI will select the first listening port that is free when it needs one. This is a problem for firewalls because they need to know beforehand precisely which ports should be allowed through. This attribute specifies that this specific port number should be used when a listening port is needed. Only one port number is necessary but that port must be free. This means that no other application on the same machine should be listening on this port.</p>	
Type:	String
Required:	No
Mutability:	Restart required

<b>maxMessages</b>	
<p>The maximum number of messages that will be returned from a call to <code>getMessagesFrom()</code> on the Session.</p> <p>This can be used to limit the amount of messages created simultaneously in order to conserve memory. This value is optional and defaults to zero. Zero indicates that there is no cap on the number of messages to return. Valid values are non-negative integers.</p>	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	0

## RoutingExampleSource

<b>id</b>
ID of configuration element
Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.test.RoutingExampleSource
<b>msgTypes</b>	
The FIX message types to process. If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required
<b>interval</b>	
Interval between sending two messages.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	5000
<b>side</b>	
Whether this processor is the sell side or the buy side.	

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	buy
Legal values:	sell   buy
<b>deliverTo</b>	
The value of the DeliverToCompID (FIX field 128) tag.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>onBehalfOf</b>	
The value of OnBehalfOfCompID (FIX field 115) tag.	
Type:	String
Required:	No
Mutability:	Reactivation required

## RulesPack

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required

description	
A human readable description of the rules pack.	
Type:	String
Required:	No
Mutability:	Reactivation required
rules	
The list of rules owned by this rules pack. (JMX API only).	
Type:	Object
Required:	No
Mutability:	Reactivation required

## SSLSocketConnection

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	No

## Configuration Reference

Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.core.session.connection.SSLSocketConnection
<b>hostname</b>	
For client connections, the network name or machine address of the server to connect to.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>port</b>	
For client connections, the port on the server to connect to.	
Type:	Integer
Required:	No
Mutability:	Reactivation required
Minimum value:	1
Maximum value:	65535
<b>listenport</b>	
For server connections, the port to listen on for incoming connections.	
If this attribute is specified, the hostname and port attributes are ignored.	
Type:	Integer
Required:	No
Mutability:	Reactivation required
Minimum value:	1
Maximum value:	65535

**noSSLv2Hello**

Controls SSL v2 Hellos.

If set to true then SSL v2 Hellos are disabled.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

**enabledProtocols**

A space-delimited list of protocols to be enabled for use on the SSL socket.

If set, the noSSLv2Hello attribute is ignored.

Typical use of this attribute is to force the use of TLS, which prevents counterparties using SSL v2 or v3 from connecting. To enforce this, the recommended value is "TLSv1 TLSv1.1 TLSv1.2". You may also wish to be more restrictive in terms of which versions of TLS to support. In that case, only specify a subset of the above three versions. If none of the specified protocols is supported, the connection can't be established.

Type:	String
Required:	No
Mutability:	Reactivation required

**incomingConnections**

A space-delimited list of IP addresses, in Classless Inter-Domain Routing (CIDR) notation, of allowable incoming connections.

Only counterparties with one of the configured IP addresses will be allowed to connect.

Type:	String
Required:	No
Mutability:	Reactivation required

**localport**

For client connections, the local port the socket is bound to.

In normal circumstances this is automatically assigned by the OS but some counter-parties require this to be specified so that the connection may pass through a firewall.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Minimum value:	1
Maximum value:	65535

**localhost**

For client connections, the hostname or IP address to which the socket should bind.

This can be set when the system has multiple IP addresses and the client socket should use a specific one. If not specified, the socket will bind to the wildcard address (0.0.0.0), leaving the Java VM to choose.

Type:	String
Required:	No
Mutability:	Reactivation required

**bindAddress**

For server connections, the hostname or IP address to which the listening socket should bind.

This can be set when the system has multiple IP addresses and the server should listen only on a specific one. If not specified, the socket will bind to the wildcard address (0.0.0.0).

Type:	String
Required:	No
Mutability:	Reactivation required

**proxyHostname**

Hostname for proxy to use when connecting to the counter-party.

Type:	String
Required:	No
Mutability:	Reactivation required

### **proxyPort**

Port for proxy. Optional.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Minimum value:	1
Maximum value:	65535

### **connectTimeout**

Timeout for connection to the other party.

The timeout is considered to be in milliseconds and a timeout of zero is interpreted as an infinite timeout. If timeout expires before connecting a `SocketTimeoutException` is thrown. Optional, default value is 0.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	0
Minimum value:	0

### **tcpNoDelay**

Controls whether Nagle's algorithm is used by this connection.



When true, Nagle's algorithm is disabled and the TCP/IP stack uses the TCP\_NO\_DELAY option. By default Nagle's algorithm is used.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### **enabledCipherSuites**

A space-delimited list of Cipher Suites for use on the SSL socket

The client and the server must agree on the specific cipher suite that is going to be used in exchanging messages. Both the client and the server must support the agreed upon cipher suite. If the client and server do not agree on a cipher suite, no connection will be made.

Type:	String
Required:	No
Mutability:	Reactivation required

## SampleExecutor

### **id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### **class**

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.test.SampleExecutor
<b>msgTypes</b>	
The FIX message types to process. If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required
<b>interval</b>	
The interval between generating two fills.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	1000

## SampleLoggingMessageProcessor

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required

<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.test.SampleLoggingMessageProcessor
<b>msgTypes</b>	
The FIX message types to process with this listener. If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required
<b>logPrefix</b>	
A string that prefixes each logged message.	
Type:	String
Required:	Yes
Mutability:	Reactivation required

## SampleMarketDataService

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes

Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.marketdata.test.- SampleMarketDataService
<b>interval</b>	
The period between market data updates from this service.	
Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	1000
Minimum value:	0
<b>latencyCheck</b>	
Enables the addition of a milli-second resolution timestamp into the Last Trade Time field so that the message latency can be checked.	
Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

**securityStatus**

Enables security status updates.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

**update**

Controls the types of updates that are generated.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	top
Legal values:	top   depth   detail

**nonStandardMarketDataFields**

Non standard market data fields.

Type:	Object
Required:	No
Mutability:	Reactivation required

## SampleMessageListener

**id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### class

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.test.SampleMessageListener

### msgTypes

The FIX message types to process with this processor. If not set, all message types will be processed.

Type:	String
Required:	No
Mutability:	Restart required

### name

A string which prefixes each logged message.

Type:	String
Required:	No
Mutability:	Reactivation required

### verboseOutput

Controls whether the message logging output is separated out by tags or all in one line.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

## SampleMessageSource

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.test.SampleMessageSource
uniqueCIOrdID	
Controls whether each message has a unique CIOrdID.	
Type:	String
Required:	No

Mutability:	Reactivation required
Default value:	true
Legal values:	true   false
<b>sequentialCLOrdID</b>	
Controls whether the value of CLOrdID contains a sequential ID starting from 1. This has no effect if uniqueCLOrdID is false.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false
<b>checkBeforeSend</b>	
Controls whether the message processor checks that the session is logged in before attempting to send a message.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false
<b>interval</b>	
The time between sending of each message.	
Type:	String
Required:	No



## Configuration Reference

Mutability:	Reactivation required
Default value:	10000
<b>targets</b>	
A comma-separated list of parties which are used to populate the TargetCompID field in consecutive messages.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>senders</b>	
A comma-separated list of parties which are used to populate the SenderCompID field in consecutive messages.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>randomOrdType</b>	
Controls whether the OrdType field (FIX tag 40) is populated with random values ranging between from 1 and 4, otherwise 40=2 is used.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

# SampleMonitor

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.test.SampleMonitor
sessionsToMonitor	
Comma separated list of sessions that will be monitored for a disconnection	
Type:	String
Required:	No
Mutability:	Reactivation required
sessionsToDisconnect	
Comma separated list of sessions that will be disconnected if its corresponding monitored sessions gets disconnected.	
Type:	String
Required:	No

Mutability:	Reactivation required
-------------	-----------------------

## Schedule

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
parentSchedule	
A reference to another schedule.	
Can be used to include a common schedule such as holiday dates.	
Type:	Reference
Required:	No
Mutability:	Reactivation required
timeZone	
Specifies the time zone of this schedule.	
Can be any value that can be parsed by the standard <a href="#">TimeZone.getTimeZone</a> method, or "ANY_TIME_ZONE", which indicates that a parent schedule should inherit the time zone of the schedule that refers to it. This special value can be used when there are multiple schedules with different time zones, and can only be used for parent schedules.	
The default is to use the current time zone of the machine the node is running on.	
Type:	String
Required:	No

Mutability:	Reactivation required
<b>includedDates</b>	
Set of dates for which schedule is applicable.	
(JMX configuration only)	
Type:	Object
Required:	No
Mutability:	Reactivation required
<b>excludedDates</b>	
Set of dates for which schedule is not applicable.	
(JMX configuration only)	
Type:	Object
Required:	No
Mutability:	Reactivation required

## ScheduledOrderCacheProcessor

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	

## Configuration Reference

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.ordercache.- ScheduledOrderCacheProcessor
<b>msgTypes</b>	
The FIX message types to process with this listener. If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required
<b>timezone</b>	
Time zone which governs the schedules	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>dataDirectory</b>	
Cache data directory	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	scheduledOrderCache

# SchedulingDefinitions

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required

## SelectByTagValue

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.messageselector.SelectByTagValue
dictionaryName	
Dictionary to use for tag name lookup.	

If not supplied, the dictionary of the enclosing session is used. If this selector is shared between many sessions then the dictionary needs to be specified, or all of the sessions that share it must have the same dictionary.

Type:	String
Required:	No
Mutability:	Reactivation required

### selection

A list of tags and their values which will form the basis of the selection (e.g. '49=Bigbroker 35=8').

Type:	String
Required:	Yes
Mutability:	Reactivation required

### delim

The delimiter that separates the tags value pairs in the selection list.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	

## SelectorReference

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes

Mutability:	Restart required
<b>refid</b>	
The ID of selector configuration this element refers to.	
Type:	Reference
Required:	Yes
Mutability:	Reactivation required

## Session

<b>class</b>	
Java class implementing the session's IParty interface.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.core.session.Party
<b>counterpartycompid</b>	
The TargetCompID (tag56) that is used when sending outbound messages on this FIX session.	
Messages received from the counterparty of this Session should also have this value in their SenderCompID field.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>counterpartysubid</b>	
The TargetSubID (tag57) that is used when sending outbound messages on this FIX session.	



Messages received from the counterparty of this Session should also (optionally) have this value in their SenderSubID field.

Type:	String
Required:	No
Mutability:	Restart required

### counterpartylocationid

The TargetLocationID (tag143) that is used when sending outbound messages on this FIX session.

Also messages received from the counterparty of this Session should also (optionally) have this value in their SenderLocationID field.

Type:	String
Required:	No
Mutability:	Restart required

### keyStore

The name of the file containing encryption keys, when using SSL.

See [SSL Encryption](#) for more detail.

Type:	String
Required:	No
Mutability:	Reactivation required

### keyStorePassword

Password to access the key store, when using SSL.

See [SSL Encryption](#) for more detail.

Type:	String
Required:	No
Mutability:	Reactivation required

Default value:	password
<b>keyPassword</b>	
Password to access the key within the key store, when using SSL.	
See <a href="#">SSL Encryption</a> for more detail.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	password
<b>needClientAuth</b>	
Specifies whether or not client authorization is required, when using SSL.	
See <a href="#">SSL Encryption</a> for more detail.	
Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false
<b>side</b>	
Only used in a FIX to FIX routing configuration and indicates whether this session is in the 'buy side' or 'sell side' routing group.	
It is only really necessary to specify this if the FIX application is doing routing or handling both buy and sell side connections.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	sell

Legal values:	buy   sell
<b>startTask</b>	
The scheduled task which is used to start FIX sessions, when using scheduling to automatically start sessions.	
See <a href="#">Scheduling</a> for more detail.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.server.task.StartTask
<b>stopTask</b>	
The scheduled task which is used to stop FIX sessions, when using scheduling to automatically stop sessions.	
See <a href="#">Scheduling</a> for more detail.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.server.task.StopTask
<b>resetTask</b>	
The scheduled task which is used to reset FIX sessions, when using scheduling to automatically reset sessions.	
See <a href="#">Scheduling</a> for more detail.	
Type:	String
Required:	No
Mutability:	Reactivation required

Default value:	com.camerontec.catalys.server.task.ResetTask
<b>trustStore</b>	
The trust store, when using SSL.	
See <a href="#">SSL Encryption</a> for more detail.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>trustStorePassword</b>	
The trust store password, when using SSL.	
See <a href="#">SSL Encryption</a> for more detail.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	password
<b>fixversion</b>	
FIX Version for this session.	
Type:	String
Required:	Yes
Mutability:	Reactivation required
Legal values:	4.0   4.1   4.2   4.3   4.4   5.0   5.0SP1   5.0SP2
<b>transportVersion</b>	
Version of transport protocol to be used when communicating over this session.	

## Configuration Reference

Type:	String
Required:	No
Mutability:	Reactivation required
Legal values:	1.1

### heartbeat

The interval (in seconds) between heartbeat messages.

Corresponds to the standard FIX HeartBtInt field. The default value of zero means no heartbeats will be sent.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	0
Minimum value:	0

### heartbeatOffset

The number of seconds to add to the heartbeat interval in order to establish the network timeout time.

On busy networks, a value greater than the default 10 seconds may improve connection stability.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	10
Minimum value:	0

### compid

The SenderCompID (tag49) that is used when sending outbound messages on this FIX session.

Should also be the TargetCompID of messages received on this session.

Type:	String
Required:	Yes
Mutability:	Restart required
<b>subid</b>	
<p>The SenderSubID (tag50) that is used when sending outbound messages on this FIX session.</p> <p>Should also (optionally) be the TargetSubID of messages received on this session.</p>	
Type:	String
Required:	No
Mutability:	Restart required
<b>locationid</b>	
<p>The SenderLocationID (tag142) that is used when sending outbound messages on this FIX session.</p> <p>Should also (optionally) be the TargetLocationID of messages received on this session.</p>	
Type:	String
Required:	No
Mutability:	Restart required
<b>autoLogoutResponse</b>	
<p>Determines whether another Logout message will be sent immediately in response to an incoming Logout message.</p> <p>According to the FIX protocol, a Logout message must be responded to with another Logout message. However, in some cases it might be necessary to delay the Logout reply. For example, an application may have more messages to send before accepting the Logout, in which case this attribute should be set to false.</p>	
Type:	Boolean
Required:	No
Mutability:	Reactivation required

Default value:	true
<b>counterpartyDetectionEnabled</b>	
Determines whether the session will advertise its own capabilities and attempt to detect the capabilities of its counterparty in order to optimize communication for this session.	
Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false
<b>description</b>	
Free text description of session.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>resendTimeout</b>	
Specifies the number of milliseconds to wait for resent messages after sending a ResendRequest.	
Requires dialectAllowResendTimeout to be set true. If no messages are received in response to the ResendRequest after the given number of milliseconds, an error is logged and the Session will skip over these missed messages to the next message received. If this value is zero, then the session will wait forever.	
Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	0
Minimum value:	0

**dialectAllowLogonResponseWithoutEncryptMethod**

Allows logon to a counterparty which does not specify the EncryptMethod (tag98) in its Logon response message.

The FIX specification requires that the EncryptMethod tag be set both in the Logon request and response. However, some FIX engines do not process this tag and do not set it in the Logon response.

With this dialect enabled the Catalys Node will accept Logon response messages which do not contain the EncryptMethod tag. However, the Catalys Node will still set the EncryptMethod tag in its Logon request.

This dialect only applies when the Catalys Node is the initiator of the FIX connection. That is, it does not modify the requirements for Logon messages accepted by the Catalys Node.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

**dialectAllowNoLengthOnRawData**

Allows the processing of RawData fields (tag96) which do not have a corresponding RawDataLength field (tag95).

The FIX specification requires that RawData field is immediately preceded by a RawDataLength field. This is necessary because the RawData field can contain binary data and that binary data may contain the standard FIX field delimiter, SOH. This means that this delimiter cannot be used reliably to terminate a raw data field.

However, some FIX implementations assume that the RawData field will not contain binary data and therefore omit the RawDataLength field. The Node will normally reject any message containing the RawData field without a RawDataLength field. If this dialect is set, however, the Node will simply use SOH (FIX field delimiter) as an indication of the end of RawData field.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false



**dialectAllowResendTimeout**

Allows the "resendTimeout" attribute of the Session to be set.

If resendTimeout triggers before the requested messages start to arrive from the FIX counter-party, then processing will skip to the next available message that has been received. (Example: the Node receives message 20 when expecting 15, it sends a ResendRequest for 15 to 19. If the resendTimeout is set to 10000 ms and message 15 does not arrive within 10 seconds, a warning is logged and processing continues with message 20.)

WARNING: Use of this dialect will lose messages when the FIX counter-party does not respond to ResendRequests within the resendTimeout window. It is designed to be used with non-FIX compliant applications that cannot be changed to conform to the FIX specification.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

**dialectDisconnectAfterResendTimeout**

Determines whether or not the session logs out and disconnects when there is no response during the resend timeout period.

This dialect is used in conjunction with "dialectAllowResendTimeout" and "resendTimeout".

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

**dialectAlwaysSendPossDupFlagAndOrigSendingTime**

Determines whether the PossDupFlag and OrigSendingTime are added to (not resent) outbound FIX messages.

PossDupFlag will have value "N" and OrigSendingTime is set to be equal to the value of SendingTime. If the message is being resent the values of PossDupFlag and OrigSendingTime will follow standard FIX specification.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

**dialectAlwaysSendResendRequest**

Determines whether to generate a ResendRequest every time a message is received with a sequence number higher than expected.

Normally, if a first message is received with a sequence number higher than expected and other messages follow sequentially following the first, only one Resend Request would be issued. With the dialect set, one will be issued for the first message and each subsequent one.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

**dialectDoNotSendTargetCompID**

Determines whether the TargetCompID (tag56) is removed from every outgoing message.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

**dialectEnableShortTestReqIDFormat**

Determines the format of the TestReqID (tag112).

If enabled, TestReqID will be sent in the format "HHmmss" where HH - hours in the day 0/24, mm - minutes, ss - seconds).

If not enabled, the format of the TestReqID is the standard date format.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

#### **dialectIgnoreIDFieldsInLogonResponse**

Determines whether the party ID fields (CompID, SubID, LocationID) that are returned in a Logon message are used.

If set, the party fields are ignored.

When a party sends a logon request, the TargetCompID, SubID and LocationID fields together define the party with whom we wish to start a FIX session. The other party must reply with a responding logon message. Normally, the SenderCompID, SubID and LocationID in that response will be the same as the Target ID values in the request. If, however, they are not the same, the Node assumes that the other party wishes to use these new IDs and will use those IDs for referring to the party for the rest of the session.

However, some FIX implementations do not correctly set the Sender IDs in a logon response. If this dialect is set, the Node will stick with the original party Target IDs irrespective of what IDs the counterparty responds with.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

#### **dialectIgnoreInputMessageSequencing**

Determines whether input messages are processed immediately even if they are out of sequence.

If this dialect is set, all input messages are processed as if their sequence numbers were correct. Note the session's input sequence number will not be updated until logout.

Type:	Boolean
Required:	No

Mutability:	Reactivation required
Default value:	false
<b>dialectInitializeCipherOncePerSession</b>	
<p>Determines whether, when using Encryption, the cipher is initialised once at the beginning of each session, or for every FIX message.</p> <p>Normally the encryption cipher should be reinitialized for each FIX message. Otherwise, any missed or corrupted messages will result in the sending party's cipher and receiving party's cipher getting "out of sync".</p>	
Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false
<b>dialectInsertCustomFieldsInHeader</b>	
<p>Configures custom fields to be sent in the header of every FIX message, in configurations where a counterparty is expecting those custom fields.</p> <p>The value of this attribute should contain a comma separated list of custom tag and value pairs. For example: dialectInsertCustomFieldsInHeader="5004=1,5006=0".</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>dialectOrigSendingTimeInGapFill</b>	
<p>Determines whether the OrigSendingTime (tag122) is added to gap fill messages.</p> <p>The field will contain the current time.</p>	
Type:	Boolean
Required:	No

Mutability:	Reactivation required
Default value:	false
<b>dialectNumericSendingTime</b>	
<p>Determines whether SendingTime (tag52) and OrigSendingTime (tag122) are sent as numeric values rather than FIX format date strings.</p> <p>If set to true, the time is represented as the number of milliseconds since midnight, January 1, 1970 UTC. This can be useful for FAST, if the SendingTime and OrigSendingTime fields are defined as integers by the FAST template.</p>	
Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false
<b>dialectProcessSkippedMessages</b>	
<p>Determines whether messages skipped as a result of a Sequence Reset or a Gap Fill should be processed if they have been received.</p> <p>For example, suppose that messages 1,2, 4 and 5 have been received. A resend request is sent for message 3. In the meantime the processing of messages 4 and 5 are held up until 3 has arrived. If the other side replies with a sequence reset or gap fill setting the new sequence number to 6, normally message 3 would be forgotten about messages 4 and 5 discarded, as instructed by the sequence reset. This is standard FIX behaviour.</p> <p>However, some FIX implementations assume that any message that are received will be processed, even if they have sent a sequence reset. This dialect instructs the FIX engine to behave in this non-standard fashion.</p>	
Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

**dialectResendRequestEndSeqNoAlwaysInfinity**

Determines the value of the EndSeqNo (tag16) in ResendRequest messages.

If set to true, the value of EndSeqNo is infinity ("999999" for FIX 4.0 and 4.1, "0" for later versions), as required by some FIX implementations (e.g. CME iLink 2.0 and SWIFTNet FIX).

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

**dialectSequenceResetToLastSeqno**

Determines whether the NewSeqNo (tag36) in a SequenceReset message is interpreted as the last sequence number received.

The FIX specification states that the NewSeqNo field in a Sequence Reset message should specify the "next sequence number immediately following the messages and/or sequence numbers being skipped". However, as required by some FIX engines, when this dialect is true, this field is interpreted as the last sequence number. The next sequence number will be NewSeqNo + 1.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

**dialectIgnoreEmptyTags**

Allows messages with fields which have a tag and not a value.

The FIX specification states that "All tags must have a value specified". By default, the Node enforces this restriction on inbound messages and will reject messages that have tags with no values.

If this dialect is enabled, this restriction will not be enforced and tags without values will be permitted.

This dialect can only be used if trustedMode="false".

Type:	Boolean
-------	---------

Required:	No
Mutability:	Reactivation required
Default value:	false
<b>dialectIgnoreTimestampFormat</b>	
<p>Allows messages with non-standard timestamp formats.</p> <p>In the FIX protocol, The SendingTime (tag52) and OrigSendingTime (tag122) fields must comply with a FIX version-specific timestamp format. The Node validates that these fields on incoming messages comply with the correct timestamp format and rejects messages that don't.</p> <p>If this dialect is enabled, the Catalys Node will not validate the format of these fields on incoming messages.</p> <p>This dialect can only be used if trustedMode="false".</p>	
Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false
<b>dialectIgnoreCompanyID</b>	
<p>Determines whether SenderCompID (tag49) and TargetCompID (tag56) are validated over the course of a FIX session.</p> <p>By default, the Catalys Node checks the consistency of the SenderCompID and TargetCompID fields.</p> <p>If this dialect is enabled, the Catalys Node will not validate these fields. Note that it is still possible for the Node to identify which session each message (except Logon) is intended for by looking at which socket the message was received on.</p> <p>This dialect can only be used if trustedMode="false".</p>	
Type:	Boolean
Required:	No
Mutability:	Reactivation required

Default value:	false
<b>dialectIgnoreTagOrdering</b>	
<p>Allow non-compliant ordering of tags in FIX message.</p> <p>By default, the Node enforces a certain ordering on the fields of incoming messages.</p> <ul style="list-style-type: none"> <li>• BeginString must be first</li> <li>• BodyLength must be second</li> <li>• MsgType must be third</li> <li>• CheckSum must be last</li> </ul> <p>This dialect can be used to disable validation of the ordering of the MsgType field on incoming messages. Note that the FIX specification mandates the order of these fields, but this dialect is provided to in order to support non-compliant engines.</p> <p>This dialect can only be used if trustedMode="false".</p>	
Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false
<b>dialectIgnoreBeginString</b>	
<p>Accept messages where FIX version given in the BeginString (tag8) does not match that configured for the session.</p> <p>According to the FIX specification, the BeginString field specifies the FIX version in use. By default, if the Node receives a message where this does not match the session's configured version then the message will be rejected.</p> <p>This dialect can be used to disable the check.</p> <p>This dialect can only be used if trustedMode="false".</p>	
Type:	Boolean
Required:	No



Mutability:	Reactivation required
Default value:	false
<b>dialectAllowNonResetLogonMessagesMidSession</b>	
<p>Accept any Logon message.</p> <p>Normally the Catalys Node does not permit Logon messages to be sent to an already logged-on session, unless the ResetSeqNumFlag (tag141) is set to "Y".</p> <p>If this dialect is enabled the Node accepts any Logon message and will reply with a Logon reply.</p>	
Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false
<b>dialectSendNextExpectedMsgSeqNumOnLogin</b>	
<p>Determines whether NextExpectedSeqNum (tag789) is added to outgoing Logon messages.</p> <p>The FIX specification allows for message recovery at login using the NextExpectedMsgSeqNum tag, the use of which is optional, and used by agreement between parties.</p> <p>As session initiator, the Catalys Node will use this tag only if this dialect is enabled. As the session acceptor, the Catalys Node will use this tag if the initial logon message from the session initiator contains it, and so the dialect is not needed.</p>	
Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false
<b>dialectHandleMissingPossDupOnGapFill</b>	
<p>Accepts SequenceReset/GapFill messages without the PossDupFlag (tag43).</p>	

By default, the Catalys Node expects SequenceReset/GapFill messages to have PossDupFlag set to "Y". The Node will terminate the session when PossDupFlag is not set in the inbound message and its sequence number is lower than expected.

If this dialect is enabled the Node will instead process the message as if it was received in sequence.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

#### **dialectDontSendPossDupFlag**

Disables the setting of PossDupFlag (tag43) in messages that have been sent previously.

By default, the Node sets PossDupFlag to "Y" in all messages that are being resent.

If this dialect is enabled the Node will not set PossDupFlag in messages.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

#### **dialectToFixResendRequestMaxGap**

If greater than zero, CameronFIX limits message gap range in ResendRequest to this value.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	0

#### **dialectCustomResendRequestMaxGap**

If true application could implement custom max gap handling.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

**dontRejectBusinessRejects**

Determines whether BusinessReject messages are rejected.

If true, business reject messages will not be rejected. This avoids creating an infinite loop in the case that both parties reject business reject messages of the other party by sending a business reject message.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

**logonPassThroughEnabled**

Determines whether Logon messages will be passed through to user logic rather than being automatically processed by the engine.

If this session is accepting FIX connections and this option is 'true', then received logon messages will be passed through. User-defined logic in, for instance, a message processor or some backend system can then decide whether to accept the logon or reject it. A logon can be accepted by sending a logon message back to the session, or rejected by sending a logout message.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

**populateLastMsgSeqNumProcessed**

Instruct the session to populate LastMsgSeqNumProcessed (tag369) in every message sent.

This tag can be used for detecting a backlog with a counterparty.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### **validateMessageTypes**

Determines whether the session validates the MsgType of incoming messages.

If true, the session will validate the MsgType of incoming messages. The MsgType must be supported by the FIX version of the session, or it must begin with "U".

Note that clients can customize the MsgTypes supported by different FIX versions using the [FIX Dictionaries](#).

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### **sendBufferSize**

Size (in bytes) of the buffer used for sending FIX messages.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	0
Minimum value:	0

### **receiveBufferSize**

Size (in bytes) of the buffer used for receiving FIX messages.

## Configuration Reference

Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	8192
Minimum value:	0

### **maxMessageLength**

Defines the maximum size (in bytes) of messages that Catalys will accept on this session.

If set to zero (the default) no validation is performed on the size of the message.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	1048576

### **useDateMilliseconds**

Determines the precision of the SendingTime (tag52) and OrigSendingTime (tag112).

If this is set to true, SendingTime and OrigSendingTime fields in outbound messages have millisecond precision.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### **useDateMicroseconds**

Determines the precision of the SendingTime (tag52) and OrigSendingTime (tag112).

If this is set to true, SendingTime and OrigSendingTime fields in outbound messages have microsecond precision.

## Configuration Reference

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false
<b>useDateNanoseconds</b>	
<p>Determines the precision of the SendingTime (tag52) and OrigSendingTime (tag112).</p> <p>If this is set to true, SendingTime and OrigSendingTime fields in outbound messages have nanosecond precision.</p>	
Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false
<b>resendRequestInfinity</b>	
<p>Specifies the value of infinity used for EndSeqNo (tag16) value in ResendRequest messages.</p> <p>From FIX 4.2 onwards, ResendRequest messages may contain an EndSeqNo of 0 which implies that all messages should be resent up until the current point. FIX 4.0 and FIX 4.1 default this value to 999999. Some counterparties may want to use their own value instead of these defaults. Setting this to the counterparty's value will ensure that if that number is received as an EndSeqNo in a Resend Request, it will be treated as infinity and not rejected as an unknown high sequence number.</p> <p>If this attribute is not set, then the default value for the session's FIX version is used.</p>	
Type:	Integer
Required:	No
Mutability:	Reactivation required
Minimum value:	0
<b>scheduledEvents</b>	

Assigns a scheduled events configuration to this Session.

Type:	Reference
Required:	No
Mutability:	Reactivation required

#### **toFIXCompressionRefId**

Assigns a Compression configuration to outgoing messages on this Session.

Type:	Reference
Required:	No
Mutability:	Reactivation required

#### **fromFIXCompressionRefId**

Assigns a Compression configuration to incoming messages on this Session.

Type:	Reference
Required:	No
Mutability:	Reactivation required

#### **fromFIXMessageFactoryRefId**

Assigns a MessageFactory configuration to outgoing messages on this Session.

Note that a Session may not contain a MessageFactory element and have this attribute.

Type:	Reference
Required:	No
Mutability:	Reactivation required

#### **toFIXMessageFactoryRefId**

Assigns a MessageFactory configuration to incoming messages on this Session.

Note that a Session may not contain a MessageFactory element and have this attribute.

Type:	Reference
Required:	No
Mutability:	Reactivation required
<b>dictionaryName</b>	
Specifies the (case sensitive) name of the dictionary used by the session for FIX message parsing and validation.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>trustedMode</b>	
<p>Determines how much validation is performed on each received message.</p> <p>If true, session messages are assumed to be essentially valid, and validation is limited. Missing and empty tags are not checked, the FIX version is not validated, inconsistent TargetCompID and SenderCompID fields are not validated, timestamping formats are not checked, and values for the PossDupFlag and PossResend flags are not mandated to be 'Y' or 'N'.</p> <p>When set to true, this overrides the following dialects: dialectIgnoreBeginString, dialectIgnoreEmptyTags, dialectIgnoreTimestampFormat, dialectIgnoreCompanyID, dialectIgnoreTagOrdering.</p>	
Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	true
<b>messageEventFactoryClass</b>	
<p>Class name of the factory that provides MessageEvent objects for each inbound message event.</p> <p>This attribute is for use with the Container only.</p>	



## Configuration Reference

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.util.messaging.- PossiblyStoredMessageEventFactory

### **printSessionLayerTimestamps**

Determines whether session layer timestamps (in nanoseconds) are logged to the console as INFO messages.

If true, timestamps are logged.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### **maximumGapQueueSize**

Maximum number of messages to store in the Gap Queue before disconnecting.

If the gap queue reaches this size, the session will be logged out from the counterparty and disconnected. Value of '-1' indicates unlimited gap queue size which may lead to out of memory exceptions.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	1000
Minimum value:	-1

### **exposeAdminMessagesToApplication**

Determines whether admin messages (ResendRequest, SequenceReset, HeartBeat, TestRequest) are passed to the application.

If true, admin messages are passed to the application.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### **maximumAppQueueSize**

Maximum number of messages to store in the App Queue.

If the app queue reaches this size, the protocol thread will be blocked and will not put more messages to the queue.

Type:	Integer
Required:	No
Mutability:	Restart required
Default value:	2000
Minimum value:	0

### **useExactFixDictionary**

Determines whether session will use dictionary of the FIX version of this session.

By default the latest FIX dictionary available is used, allowing the session to parse FIX messages containing all known FIX fields. If you would prefer to use the FIX dictionary corresponding to this session's FIX version then set this attribute to true. You might need to do this if you use custom fields in your messages that conflict with fields defined by a later version of the FIX protocol.

Type:	Boolean
Required:	No
Mutability:	Reactivation required

Default value:	false
<b>maximumInboundMessageRate</b>	
<p>The maximum number of inbound messages to read from the socket per second.</p> <p>By setting this attribute, inbound message throttling is enabled. Throttling is applied prior to reading messages from the TCP socket. Throttled messages are not buffered or queued in the application; rather, when a sender exceeds the configured rate, messages will accumulate in the inbound TCP buffer. Note this is not supported when using multiplexingMode.</p>	
Type:	Integer
Required:	No
Mutability:	Reactivation required
Minimum value:	1
Maximum value:	100000
<b>resendThrottleInterval</b>	
<p>The interval over which to count the number of allowable messages that are resent as a reply to ResendRequest message.</p> <p>This attribute is used only when the rate of the resent messages needs to be throttled. This interval is specified in seconds. When it is 0 resend throttling is disabled.</p>	
Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	0
Minimum value:	0
<b>resendThrottleMessagesPerInterval</b>	
<p>The number of messages, that are resent as a reply to ResendRequest message, allowed in the given interval.</p>	

This attribute is used only when the rate of the resent messages needs to be throttled. It is used with `resendInterval` attribute. When it is 0 resend throttling is disabled.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	0
Minimum value:	0

## SessionManager

### class

Java run-time class of the session manager.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.server.sessionmanager.SessionManager

### customAttributes

Custom attributes.

Type:	Object
Required:	No
Mutability:	Restart required
Default value:	null

### acceptConnectionsOutsideSchedule

Controls the behaviour of the server when a party attempts to connect when it is not scheduled.

If true, the party can connect. If false the connection attempts will be ignored and message processors will never see any messages from the party which is trying to connect, since the connection is dropped before a logon message is sent.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	true

### **checkInterval**

The time (in milliseconds) between periodic session status checks.

If the session is disconnected or becomes logged out, the session manager will attempt to reestablish the session after this time.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	10000
Minimum value:	0

### **reconnectDelay**

The delay (in milliseconds) between the session manager failing to create a session and the subsequent retry.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	10000
Minimum value:	0

### **maxLogonAttempts**

The maximum number of logon attempts that are allowed before no further attempts are made.

By default, there is no limit on the number of allowable logon attempts.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	0
Minimum value:	0

### **stopReconnectOnSeqNumTooLow**

Controls whether the session manager will stop making outbound logon attempts if a logon fails because the sequence number is too low.

By default, the session manager will keep attempting to logon even if the sequence number was too low.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### **clientConnectionManager**

Class responsible for maintaining connections with a counterparty.

This class must implement the `com.camerontec.catalys.server.sessionmanager.IClientConnectionManager` interface

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	<code>com.camerontec.catalys.server.sessionmanager.-DefaultClientConnectionManager</code>

**connectOnActivation**

Determines whether the session manager will automatically attempt to initiate a connection with counterparties when it is activated.

The session manager is activated once at startup and then only again if it has been manually deactivated and reactivated.

Note that, even when this attribute is true, a connection attempt will not be made if the session has been blocked or if scheduling is being used and the session is not scheduled to be active.

If this attribute is false, connections need to be initiated manually.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	true

**connectOnApplicationLogon**

Determines whether the session manager will attempt to initiate a connection with counterparties only when the application sends a logon.

If true, the session manager will not automatically attempt to initiate a connection with counterparties when it is activated. The connection will be initiated when the application sends a Logon message. This attribute is only relevant for client connection.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

**logAllConnectionAttempts**

Determines whether each failed connection attempt is logged, or just the first one.

When a client connection fails to connect to a server counterparty it will normally retry automatically until it succeeds. By default the session manager will only log the first of these attempts.

## Configuration Reference

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### **resetOnLogon**

Determines whether an automatic sequence number reset occurs with each logon

If true, the sequence number is reset and the logon message will be sent with tag141 (ResetSeqNumFlag) set to Y. This is not standard FIX, but some systems require it.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### **updateMessageHistory**

Controls whether or not the message history is updated as messages pass through the chain of message processors.

If true, each message processor adds a reference to itself to the message's history list as it processes each message. The message history list can be retrieved by calling the [getHistory](#) method.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### **passwordStoreServiceName**

Password store service name for client connection manager to use.

This attribute is not used by the default client connection manager, however it is available for use by custom client connection managers.



Type:	String
Required:	No
Mutability:	Reactivation required
<b>username</b>	
<p>Username for client connection manager.</p> <p>This attribute is ignored if used with the default client connection manager, however it is available for use by custom client connection managers. The <code>com.camerontec.catalys.server.test.UserPassClientConnectionManager</code> populates the Username field (FIX tag 553) of the Logon message if the FIX session is configured to be a client.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>password</b>	
<p>Password for client connection manager.</p> <p>This attribute is ignored is used with the default client connection manager, however it is available for use by custom client connection managers. The <code>com.camerontec.catalys.server.test.UserPassClientConnectionManager</code> populates the Password field (FIX tag 554) of the Logon messages if the FIX session is configured to be a client.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>executeMissedScheduledResetTaskOnStartUp</b>	
<p>Determines whether an automatic sequence number reset occurs if the last scheduled reset task was not executed.</p> <p>If true, the session manager will reset the sequence numbers on activation. The attribute will help in situations where the Node has been down during the time the Session's reset task was to be executed. With this set to true, when the Node is started the session will reset its sequence numbers before attempting connections.</p>	

## Configuration Reference

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

### **excludeFromMultiplexing**

Determines whether the session will be excluded from multiplexing.

To be used if a session needs different behaviour from application default. If true, this session will not run in multiplexing mode even if configured on application level.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

## SessionReference

### **id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### **class**

Java class of this element's connection point Runtime objects.

Type:	String
Required:	No

Mutability:	Reactivation required
Default value:	com.camerontec.catalys.core.session.ConnectionPoint
<b>counterpartycompid</b>	
<p>The TargetCompID (tag56) that is used when sending outbound messages on this FIX session.</p> <p>Messages received from the counterparty of this Session should also have this value in their SenderCompID field.</p>	
Type:	String
Required:	No
Mutability:	Restart required
<b>counterpartysubid</b>	
<p>The TargetSubID (tag57) that is used when sending outbound messages on this FIX session.</p> <p>Messages received from the counterparty of this Session should also (optionally) have this value in their SenderSubID field.</p>	
Type:	String
Required:	No
Mutability:	Restart required
<b>counterpartylocationid</b>	
<p>The TargetLocationID (tag143) that is used when sending outbound messages on this FIX session.</p> <p>Also messages received from the counterparty of this Session should also (optionally) have this value in their SenderLocationID field.</p>	
Type:	String
Required:	No
Mutability:	Restart required
<b>compid</b>	

The SenderCompID (tag49) that is used when sending outbound messages on this FIX session.

Should also be the TargetCompID of messages received on this session.

Type:	String
Required:	No
Mutability:	Restart required

### **subid**

The SenderSubID (tag50) that is used when sending outbound messages on this FIX session.

Should also (optionally) be the TargetSubID of messages received on this session.

Type:	String
Required:	No
Mutability:	Restart required

### **locationid**

The SenderLocationID (tag142) that is used when sending outbound messages on this FIX session.

Should also (optionally) be the TargetLocationID of messages received on this session.

Type:	String
Required:	No
Mutability:	Restart required

## SocketAdapter

### **id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
-------	--------

Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.socket.SocketAdapter
<b>msgTypes</b>	
The FIX message types to process with this listener. If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required
<b>port</b>	
Port on which the Socket Adapter listens for a client connection.	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	5005
<b>ackTimeout</b>	
The amount of time (in milliseconds) the server should wait for acknowledgment before it considers the connection dead and disconnects.	

If the Socket Adapter has sent a RECV\_MESSAGE and has not received a SEND\_ACK within ackTimeout then it will disconnect. If this value is set to 0 then the Socket Adapter does not wait for acknowledgment.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	0

### multiSession

Controls whether the Socket Adapter is listening across multiple sessions or just a single session.

When this attribute is set, the Socket Adapter can be used to send and receive to/from multiple sessions, as opposed to just having a single socket client per Socket Adapter. When this attribute is set, each outbound message must contain a valid TargetCompID, TargetSubID, TargetLocationID combination. This will allow the Socket Adapter to route each message to the appropriate session. When it is not set, outbound messages do not need targets, as each message has only one possible destination.

Note: When the multiSession attribute is set to true it is not possible to use different 'to FIX' Message Factories for different Socket Adapter instances, they must all be the same (or left at the default value). If different Message Factories are configured there is no guarantee which one will be used.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	false
Legal values:	true   false

### alwaysUseMsgId

Controls whether acknowledgements from the socket client are expected to only include the message id from the original message, or the body of the original message.

If true, acknowledgements should include just the message id, otherwise they should include the body of the message.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

### **addDataToAckMsg**

Configures the contents of the Data field of the ack message.

If set to none, the data field of the ack message will be empty. If set to seqnum the the data field of the ack message will contain the outbound sequence number assigned to the message (if known).

In the special case of SourceMessageBuffer message processors, the sequence number will not be set if the message was buffered because the session was not available to send the message directly.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	seqnum
Legal values:	none   seqnum

### **largeMessages**

Controls the value of the MessageLength field.

If true, the MessageLength field will be 9 bytes. If false it will be 6 bytes. This needs to be set to true to allow messages of size greater than 999999 bytes to be transmitted.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false

Legal values:	true   false
---------------	--------------

<b>fullMessageAcks</b>	
Controls whether the Socket Adapter will acknowledge the client when it has sent a message to FIX. If true, the acknowledgement is of message type RECV_ACK_MSG, and contains the full message.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

<b>keepAlive</b>	
Controls whether the underlying socket is kept open even if messages are sent only sporadically. If true, SO_KEEPALIVE is set on the underlying socket which will cause the socket to remain open even if no messages are sent or received in a long period of time. This is useful for sessions that should remain up during the day with only a few messages sent or received per day.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

<b>transformer</b>	
Class that converts inbound/outbound FIX messages and events to and from socket messages. Must implement <a href="#">IMessageTransformer</a> .  The Socket Adapter can take a nested Properties element, whose attributes will be passed through to the transformer. The default transformer uses the nested property 'partyDelimiter'. The value of this property is used to separate the parties in the	



session ID. For example, setting partyDelimiter="~~~" would give a session ID of TheirCompID.TheirSubID.TheirLocationID~~~OurCompID.OurSubID.OurLocationID. The partyDelimiter defaults to "\_". The session ID is included in message event messages (RECV\_CONNECTION\_STATUS, RECV\_EXCEPTION, RECV\_RESET\_MSG).

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.server.adapter.socket.- SocketAdapterRawTransformer

### autoForwarding

Controls whether messages are automatically forwarded to other listeners in the chain.

If true, messages are automatically forwarded.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

## SocketConnection

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

**class**

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.core.session.connection.SocketConnection

**hostname**

For Client connections, the network name or machine address of the server to connect to.

Type:	String
Required:	No
Mutability:	Reactivation required

**port**

For Client connections, the port on the server to connect to.

Type:	Integer
Required:	No
Mutability:	Reactivation required
Minimum value:	1
Maximum value:	65535

**localhost**

For Client connections, the hostname or IP address to which the socket should bind.

This can be set when the system has multiple IP addresses and the client socket should use a specific one. If not specified, the socket will bind to the wildcard address (0.0.0.0), leaving the Java VM to choose.

Type:	String
Required:	No
Mutability:	Reactivation required
<b>localport</b>	
<p>For Client connections, the local port the socket is bound to.</p> <p>In normal circumstances this is automatically assigned by the OS but some counter-parties require this to be specified so that the connection may pass through a firewall.</p>	
Type:	Integer
Required:	No
Mutability:	Reactivation required
Minimum value:	1
Maximum value:	65535
<b>listenport</b>	
<p>For Server connections, the port to listen on for incoming connections.</p> <p>If this attribute is specified, the hostname and port attributes are ignored.</p>	
Type:	Integer
Required:	No
Mutability:	Reactivation required
Minimum value:	1
Maximum value:	65535
<b>bindAddress</b>	
<p>For Server connections, the hostname or IP address to which the listening socket should bind.</p> <p>This can be set when the system has multiple IP addresses and the server should listen only on a specific one. If not specified, the socket will bind to the wildcard address (0.0.0.0).</p>	

Type:	String
Required:	No
Mutability:	Reactivation required
<b>receiveBufferSize</b>	
Preferred socket receive buffer size in bytes.	
This value provides a hint to the OS. The actual size is determined by the TCP/IP stack of the particular operating system.	
Type:	Integer
Required:	No
Mutability:	Reactivation required
Minimum value:	0
<b>connectionSendBufferSize</b>	
Preferred socket send buffer in bytes.	
This value provides a hint to the OS. The actual size is determined by the TCP/IP stack of the particular operating system.	
Type:	Integer
Required:	No
Mutability:	Reactivation required
Minimum value:	0
<b>incomingConnections</b>	
A space-delimited list of IP addresses, in Classless Inter-Domain Routing (CIDR) notation, of allowable incoming connections.	
Only counterparties with one of the configured IP addresses will be allowed to connect.	
Type:	String

## Configuration Reference

Required:	No
Mutability:	Reactivation required
<b>proxyHostname</b>	
Hostname for proxy to use when connecting to the counter-party.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>proxyPort</b>	
Port for proxy to use when connecting to the counter-party.	
Type:	Integer
Required:	No
Mutability:	Reactivation required
Minimum value:	1
Maximum value:	65535
<b>connectTimeout</b>	
Timeout in milliseconds for connection to the counter party.	
If this value is zero then the connection will not timeout. If this the timer expires before the connection has been made, a <code>SocketTimeoutException</code> is thrown.	
Type:	Integer
Required:	No
Mutability:	Reactivation required
Default value:	0
Minimum value:	0

**tcpNoDelay**

Controls whether Nagle's algorithm is used by this connection.

When true, Nagle's algorithm is disabled and the TCP/IP stack uses the TCP\_NO\_DELAY option. By default Nagle's algorithm is used.

Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false

## SolaceEndpoint

**id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

**class**

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.solace.SolaceEndpoint

**transformer**

Class which transforms FIX messages into Solace messages.

Must implement [IMessageTransformer](#).

Note that since this adapter sets the storeMessages attribute to false to enable message reuse, transformers which are intended to be used with this adapter may not store any references to the FIX message object.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.server.adapter.solace.SolaceRawTransformer

### msgTypes

The FIX message types to process with this listener.

If not set, all message types will be processed.

Type:	String
Required:	No
Mutability:	Restart required

### host

Comma-separated list of Solace appliance host names or IP addresses, including the port if not using the default.

Type:	String
Required:	Yes
Mutability:	Reactivation required

### bindAddress

Hostname or IP address of local network interface on which to bind.

The Solace API will communicate with the appliance from this address.

Type:	String
-------	--------

## Configuration Reference

Required:	No
Mutability:	Reactivation required
<b>username</b>	
Username used for authentication with the appliance.	
Type:	String
Required:	Yes
Mutability:	Reactivation required
<b>password</b>	
Password used for authentication with the appliance.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>vpnName</b>	
The name of the message VPN to join when connecting to the appliance.	
See Solace documentation for default value.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>deliveryMode</b>	
How messages are delivered to/from the appliance.	
Type:	String
Required:	No



Mutability:	Restart required
Default value:	guaranteed
Legal values:	direct   guaranteed
<b>messageTimeToLive</b>	
TTL value (in milliseconds) of each published message.	
The default value is 0 (no expiration).	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	0
<b>dmqEligible</b>	
If true, each message published to the destinationTopic/Queue will be DMQ (Dead Message Queue) eligible.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false
<b>destinationQueue</b>	
Name of the destination queue where transformed FIX message are placed.	
If defined, the destinationTopic parameter is ignored.	
Type:	String
Required:	No

Mutability:	Reactivation required
<b>destinationTopic</b>	
Name of the destination topic to where transformed FIX messages are published.	
Ignored if the destinationQueue parameter is defined.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>topicResolverClass</b>	
Class which resolves the destination topic of each inbound FIX message.	
The class must implement <a href="#">ISolaceTopicResolver</a> .	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>jcsmpPropertiesFile</b>	
Path to file containing JCSMP session properties. Can be on the filesystem (takes precedence) or in the classpath.	
Type:	String
Required:	No
Mutability:	Restart required
<b>replyToTopic</b>	
If configured, the replyTo of each published message will be set to this value.	
Type:	String
Required:	No

Mutability:	Reactivation required
-------------	-----------------------

<b>replyToTopicResolver</b>	
Class which resolves the replyTo topics for each inbound FIX message.	
The class must implement <a href="#">ISolaceTopicResolver</a> .	
Type:	String
Required:	No
Mutability:	Reactivation required

<b>publishWindowTimeout</b>	
Time in milliseconds to wait for the publishing window to open.	
Supported only when <code>enablePersistence</code> is true. When attempting to publish an inbound FIX message, if the persisted publishing window does not open within the specified time, the inbound message is rejected.	
The value must be greater than or equal to 2000, which corresponds to 2 * [JCSMP API ack flush timer].	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	2000

<b>sourceQueue</b>	
Name of the queue from where Solace messages are taken.	
Supported only in guaranteed mode.	
Type:	String
Required:	No
Mutability:	Reactivation required

**sourceTopic**

Name of the topic on which to subscribe for Solace messages.

Supported only in direct mode.

Type:	String
Required:	No
Mutability:	Reactivation required

**errorQueue**

If a Solace message cannot be delivered to FIX, the message is moved to this queue.

If errorTopic/errorQueue/errorDMQ are all undefined, the message either remains unacknowledged (in guaranteed mode) or is dropped (in direct mode).

Type:	String
Required:	No
Mutability:	Reactivation required

**errorTopic**

If a Solace message cannot be delivered to FIX, the message is published to this topic.

If errorTopic/errorQueue/errorDMQ are all undefined, the message either remains unacknowledged (in guaranteed mode) or is dropped (in direct mode).

Type:	String
Required:	No
Mutability:	Reactivation required

**errorTopicResolver**

Class that resolves the destination topic for error messages.

Must implement [ISolaceErrorTopicResolver](#).

Type:	String
-------	--------

Required:	No
Mutability:	Reactivation required
<b>errorDMQ</b>	
<p>If true, undeliverable messages are moved to the Dead Message Queue by the appliance.</p> <p>Cannot be defined if either <code>errorTopic</code> or <code>errorQueue</code> is defined. Supported only in guaranteed mode. If <code>errorTopic/errorQueue/errorDMQ</code> are all undefined, the message either remains unacknowledged (in guaranteed mode) or is dropped (in direct mode).</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false
<b>lastResortAction</b>	
<p>Action to perform after a consumed message cannot be delivered to FIX and the subsequent error handling fails.</p> <p>This attribute is supported only when <code>errorTopic</code> or <code>errorQueue</code> is defined. <code>logAndAck</code> is supported only in guaranteed mode.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	log
Legal values:	log   logAndAck
<b>reuseFIXMessage</b>	
<p>If true, the same <code>IFIXMessage</code> instance will be reused when transforming and firing each Solace message.</p>	

If false, a new instance will be created per Solace message.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

### **routingMetadataStartTag**

Starting tag number of the 8 routing metadata fields.

Routing metadata is added to messages consumed from a Solace source. The metadata is used to properly route rejects from a publishing session back to a consuming session. The default value is 6000, which means tags 6000 through 6007 inclusive will be used. The value of this attribute should be the same on all SolaceEndpoints that reside in the same routing domain.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	6000

### **messageIdResolver**

Class which resolves the application message ID for each published FIX message.

The class must implement [ISolaceMessageIdResolver](#).

Type:	String
Required:	No
Mutability:	Reactivation required

### **clearComplds**

If true, ComplIDs and BeginString in consumed messages are cleared and overwritten with the consuming session's ComplIDs.

Rules engine can be used to update CompIDs after consuming messages from the Solace appliance. Messages should have correct compids before routing.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

## SolaceMessageListener

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### class

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.solace.SolaceMessageListener

### transformer

Class which transforms FIX messages into Solace messages.

Must implement [IMessageTransformer](#).

Note that since this adapter sets the `storeMessages` attribute to `false` to enable message reuse, transformers which are intended to be used with this adapter may not store any references to the FIX message object.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	<code>com.camerontec.catalys.server.adapter.solace.SolaceRawTransformer</code>

### **msgTypes**

The FIX message types to process with this listener.

If not set, all message types will be processed.

Type:	String
Required:	No
Mutability:	Restart required

### **host**

Comma-separated list of Solace appliance host names or IP addresses, including the port if not using the default.

Type:	String
Required:	Yes
Mutability:	Reactivation required

### **bindAddress**

Hostname or IP address of local network interface on which to bind.

The Solace API will communicate with the appliance from this address.

Type:	String
Required:	No



Mutability:	Reactivation required
<b>username</b>	
Username used for authentication with the appliance.	
Type:	String
Required:	Yes
Mutability:	Reactivation required
<b>password</b>	
Password used for authentication with the appliance.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>vpnName</b>	
The name of the message VPN to join when connecting to the appliance.	
See Solace documentation for default value.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>deliveryMode</b>	
How messages are delivered to/from the appliance.	
Type:	String
Required:	No
Mutability:	Restart required

Default value:	guaranteed
Legal values:	direct   guaranteed
<b>messageTimeToLive</b>	
TTL value (in milliseconds) of each published message.	
The default value is 0 (no expiration).	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	0
<b>dmqEligible</b>	
If true, each message published to the destinationTopic/Queue will be DMQ (Dead Message Queue) eligible.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false
<b>destinationQueue</b>	
Name of the destination queue where transformed FIX message are placed.	
If defined, the destinationTopic parameter is ignored.	
Type:	String
Required:	No
Mutability:	Reactivation required

**destinationTopic**

Name of the destination topic to where transformed FIX messages are published.

Ignored if the destinationQueue parameter is defined.

Type:	String
Required:	No
Mutability:	Reactivation required

**topicResolverClass**

Class which resolves the destination topic of each inbound FIX message.

The class must implement [ISolaceTopicResolver](#).

Type:	String
Required:	No
Mutability:	Reactivation required

**jcsmpPropertiesFile**

Path to file containing JCSMP session properties. Can be on the filesystem (takes precedence) or in the classpath.

Type:	String
Required:	No
Mutability:	Restart required

**replyToTopic**

If configured, the replyTo of each published message will be set to this value.

Type:	String
Required:	No
Mutability:	Reactivation required

**replyToTopicResolver**

Class which resolves the replyTo topics for each inbound FIX message.

The class must implement [ISolaceTopicResolver](#).

Type:	String
Required:	No
Mutability:	Reactivation required

**enablePersistence**

Persists messages to disk until an acknowledgement is received from the appliance.

If the Solace connection is severed, upon reconnection any unacknowledged messages will be republished and flagged with PossResend=Y. The size of the persistence window is equal to the JCSMP PUB\_ACK\_WINDOW\_SIZE property. The persistence files are automatically cleared and compacted when the session is reset.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	false
Legal values:	true   false

**publishWindowTimeout**

Time in milliseconds to wait for the publishing window to open.

Supported only when `enablePersistence` is true. When attempting to publish an inbound FIX message, if the persisted publishing window does not open within the specified time, the inbound message is rejected.

The value must be greater than or equal to 2000, which corresponds to 2 \* [JCSMP API ack flush timer].

Type:	String
Required:	No

Mutability:	Restart required
Default value:	2000
<b>messageIdResolver</b>	
Class which resolves the application message ID for each published FIX message.	
The class must implement <a href="#">ISolaceMessageIdResolver</a> .	
Type:	String
Required:	No
Mutability:	Reactivation required

## SolaceMessageSource

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.solace.SolaceMessageSource
<b>transformer</b>	

Class which transforms Solace messages into FIX messages.

Must implement [ISolaceMessageTransformer](#).

Note also that since this adapter sets the storeMessages attribute to false to enable message reuse, transformers which are intended to be used with this adapter may not store any references to the FIX message object.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.server.adapter.solace.SolaceRawTransformer

### host

Comma-separated list of Solace appliance host names or IP addresses, including the port if not using the default.

Type:	String
Required:	Yes
Mutability:	Reactivation required

### bindAddress

Hostname or IP address of local network interface on which to bind.

The Solace API will communicate with the appliance from this address.

Type:	String
Required:	No
Mutability:	Reactivation required

### username

Username used for authentication with the appliance.

Type:	String
-------	--------

## Configuration Reference

Required:	Yes
Mutability:	Reactivation required
<b>password</b>	
Password used for authentication with the appliance.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>vpnName</b>	
The name of the message VPN to join when connecting to the appliance.	
See Solace documentation for default value.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>deliveryMode</b>	
How messages are delivered to/from the appliance.	
In guaranteed mode, sourceQueue must be defined and sourceTopic is not supported.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	guaranteed
Legal values:	direct   guaranteed
<b>sourceQueue</b>	
Name of the queue from where Solace messages are taken.	

Supported only in guaranteed mode.

Type:	String
Required:	No
Mutability:	Reactivation required

### sourceTopic

Name of the topic on which to subscribe for Solace messages.

Supported only in direct mode.

Type:	String
Required:	No
Mutability:	Reactivation required

### errorQueue

If a Solace message cannot be delivered to FIX, the message is moved to this queue.

If errorTopic/errorQueue/errorDMQ are all undefined, the message either remains unacknowledged (in guaranteed mode) or is dropped (in direct mode).

Type:	String
Required:	No
Mutability:	Reactivation required

### errorTopic

If a Solace message cannot be delivered to FIX, the message is published to this topic.

If errorTopic/errorQueue/errorDMQ are all undefined, the message either remains unacknowledged (in guaranteed mode) or is dropped (in direct mode).

Type:	String
Required:	No
Mutability:	Reactivation required



**errorTopicResolver**

Class that resolves the destination topic for error messages.

Must implement [ISolaceErrorTopicResolver](#).

Type:	String
Required:	No
Mutability:	Reactivation required

**errorDMQ**

If true, undeliverable messages are moved to the Dead Message Queue by the appliance.

Cannot be defined if either errorTopic or errorQueue is defined. Supported only in guaranteed mode. If errorTopic/errorQueue/errorDMQ are all undefined, the message either remains unacknowledged (in guaranteed mode) or is dropped (in direct mode).

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

**jcsmpPropertiesFile**

Path to file containing JCSMP session properties.

Can be on the filesystem (takes precedence) or in the classpath.

Type:	String
Required:	No
Mutability:	Reactivation required

**reuseFIXMessage**

If true, the same IFIXMessage instance will be reused when transforming and firing each Solace message.

If false, a new instance will be created per Solace message.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

### **waitForLogon**

If true, message consumption will not commence until the corresponding FIX session has logged on.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

### **clearComplds**

If true, ComplIDs in consumed messages are cleared and overwritten with the consuming session's ComplIDs.

This attribute is not supported when multiple sessions share the same queue.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

**lastResortAction**

Action to perform after a consumed message cannot be delivered to FIX and the subsequent error handling fails.

This attribute is supported only when `errorTopic` or `errorQueue` is defined. `logAndAck` is supported only in guaranteed mode.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	log
Legal values:	log   logAndAck

## SoupTCPSessionService

**id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

**class**

Java class of the Runtime object corresponding to this configuration object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.soupsservice.SoupTCPSessionService

<b>username</b>	
user name	
Type:	String
Required:	Yes
Mutability:	Reactivation required
<b>passwordStoreServiceName</b>	
password	
Type:	String
Required:	Yes
Mutability:	Reactivation required
<b>addresses</b>	
addresses	
Type:	String
Required:	Yes
Mutability:	Reactivation required
<b>version</b>	
bin or ascii	
Type:	String
Required:	Yes
Mutability:	Reactivation required
Legal values:	bin   ascii
<b>sequencePersisterClass</b>	

Soup session sequence persister class

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.server.soup.service.- SoupSessionSequencePersister

### **soupPacketFactoryClass**

Soup packet factory class

Type:	String
Required:	Yes
Mutability:	Reactivation required

## SourceMessageBuffer

### **id**

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### **class**

Java class of this element's Runtime object.

Type:	String
Required:	No

Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.storeandforward.SourceMessageBuffer
<b>directoryName</b>	
<p>The name of the directory that is used to hold messages that have been held by the buffer.</p> <p>The name can be an absolute path or a relative one. The directory is automatically cleared when the buffer becomes empty so there is no maintenance required.</p>	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>clearOnReset</b>	
<p>Indicates whether buffers should clear out all messages when the sequence numbers of its associated session are reset.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false
<b>clearOnEOD</b>	
<p>Indicates whether buffers should clear out all messages when the end of day procedure is performed on the session.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false

Legal values:	true   false
<b>alwaysBuffer</b>	
<p>Indicates that the message buffer should always store all messages and deliver them asynchronously, regardless of whether synchronous delivery would have been successful or not.</p> <p>If false then buffering will only be triggered if message delivery fails. Only then will the message buffer processor insert the failed messages in the buffer. Another independent thread, the retry thread, polls the buffer every retryInterval and tries to deliver the messages asynchronously. Once the retry thread delivers all messages then messages are not buffered anymore.</p> <p>If true then the message buffer processor will always place all messages in the buffer. Another independent thread, the delivery thread, delivers the messages as soon as they are placed in the buffer. When the buffer is empty the delivery thread blocks waiting for new message(s) to be inserted in the buffer. If the delivery thread cannot deliver the message(s) then it sleeps an amount of time determined by retryInterval.</p>	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	false
Legal values:	true   false
<b>rejectIfNotSelected</b>	
<p>Indicates what to do with received messages which are not selected while buffering.</p> <p>If true, unselected messages received while buffering cause an exception to be thrown (i.e. they are rejected). If false, they are ignored.</p>	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

**reportSizeAttribute**

Name of attribute used to report changes to the number of messages held in the buffer.

If set, an attribute of that name will be used to store the number of messages in the buffer. Whenever the value of that attribute changes, an `AttributeChangeEvent` is fired which other objects can listen to by calling `addAttributeChangeListener`.

If not set then changes to the buffer size are not reported.

Type:	String
Required:	No
Mutability:	Reactivation required

**retryInterval**

The number of milliseconds between send retry attempts.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	10000

**selectorRefID**

Reference to the ID of a configured selector, allowing control over which messages are buffered.

Messages received while buffering which are not selected are rejected by default (see the `rejectIfNotSelected` attribute).

If not set then all messages are buffered.

Type:	Reference
Required:	No
Mutability:	Restart required

**compactWhenEmpty**



Indicates whether buffer's persistent collection should be compacted whenever the buffer's size reaches 0.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

## SourceMessageThrottle

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required

### class

Java class of this element's Runtime object.

Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.storeandforward.SourceMessageThrottle

### directoryName

The name of the directory that is used to hold messages that have been held by the throttle.

The name can be an absolute path or a relative one. The directory is automatically cleared when the throttle becomes empty so there is no maintenance required.

Type:	String
Required:	Yes
Mutability:	Restart required

### **selectorRefID**

The ID of a configured selector, which controls which messages are buffered.

Messages received while throttling which are not selected are rejected by default (see the `rejectIfNotSelected` attribute).

If not set then all messages are buffered.

Type:	Reference
Required:	No
Mutability:	Restart required

### **rejectIfNotSelected**

Indicates what to do with received messages which are not selected while buffering.

If true, unselected messages received while buffering cause an exception to be thrown (i.e. they are rejected). If false, they are ignored.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	true
Legal values:	true   false

### **interval**

The interval over which to count the number of allowable messages.

## Configuration Reference

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	1
<b>intervalTimeUnit</b>	
The time unit for the interval attribute. Allowed values are either 's' for seconds or 'ms' for milliseconds.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	s
Legal values:	s   ms
<b>messagesPerInterval</b>	
The number of messages to allow in the given interval.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	10
<b>compactWhenEmpty</b>	
Indicates whether throttle's persistent collection should be compacted whenever the throttle's size reaches 0.	
Type:	String
Required:	No
Mutability:	Reactivation required

Default value:	true
Legal values:	true   false

## StandAloneGenericSelector

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	Yes
Mutability:	Reactivation required

## StandAloneMsgTypeSelector

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required

<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.messageselector.MsgTypeSelector
<b>includeMsgTypes</b>	
A list of message types to include (e.g., 'D 8 G').	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>excludeMsgTypes</b>	
A list of message types to exclude (e.g., 'D 8 G').	
Type:	String
Required:	No
Mutability:	Reactivation required

## StandAloneSelectByTagValue

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes

Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.messageselector.SelectByTagValue
<b>dictionaryName</b>	
Dictionary to use for tag name lookup.	
If not supplied, the dictionary of the enclosing session is used. If this selector is shared between many sessions then the dictionary needs to be specified, or all of the sessions that share it must have the same dictionary.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>selection</b>	
A list of tags and their values which will form the basis of the selection (e.g., '49=Bigbroker 35=8').	
Type:	String
Required:	Yes
Mutability:	Reactivation required
<b>delim</b>	
The delimiter that separates the tags value pairs in the selection list.	
Type:	String

Required:	No
Mutability:	Reactivation required
Default value:	

## TIBRVMessageListener

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.tibrv.TIBRVProducer
msgTypes	
The FIX message types to process with this listener. If not set, all message types will be processed.	
Type:	String
Required:	No
Mutability:	Restart required
autoForwarding	

Controls whether messages are automatically forwarded to other listeners in the chain.

If true, messages are automatically forwarded.

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	false
Legal values:	true   false

### subject

The subject on which to publish outbound messages or from which to read inbound messages.

Type:	String
Required:	Yes
Mutability:	Reactivation required

### transport.type

Identifies the TIBRV transport type to use.

Rendezvous offers 4 types:

- RVD (daemon)
- RVA (agent)
- CM (certified messaging)
- CMQueue (distributed certified messaging)

Type:	String
Required:	No
Mutability:	Restart required
Default value:	RVD



Legal values:	RVD   RVA   CM   CMQ
<b>impl</b>	
The implementation type for TIBRV.	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	NATIVE
Legal values:	NATIVE   JAVA   SELECT
<b>reconnect.delay</b>	
Milliseconds to wait before reconnecting after a disconnection.	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	10000
<b>rvd.service</b>	
Service name for RVD transport.	
If none is specified, the TIBRV default is used.	
Type:	String
Required:	No
Mutability:	Restart required
<b>rvd.network</b>	
Network name for RVD transport.	
If none is specified, the TIBRV default is used.	

Type:	String
Required:	No
Mutability:	Restart required
<b>rvd.daemon</b>	
<p>Daemon location for RVD transport.</p> <p>If none is specified, the TIBRV default is used.</p>	
Type:	String
Required:	No
Mutability:	Restart required
<b>rva.host.name</b>	
<p>Host name for RVA transport.</p> <p>If none is specified, the TIBRV default is used.</p>	
Type:	String
Required:	No
Mutability:	Restart required
<b>rva.port</b>	
<p>Port number for RVA transport.</p> <p>If none is specified, the TIBRV default is used.</p>	
Type:	String
Required:	No
Mutability:	Restart required
<b>rva.tunnel.mode</b>	
<p>Tunnel mode for RVA transport.</p>	

If none is specified, the TIBRV default is used.

Type:	String
Required:	No
Mutability:	Restart required

#### **cm.cm.name**

CM name for CM transport.

If none is specified, the TIBRV default is used.

Type:	String
Required:	No
Mutability:	Restart required

#### **cm.request.old**

Determines whether unacknowledged messages will be re-requested from senders for CM transport.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	false
Legal values:	true   false

#### **cm.ledger.name**

The CM ledger file for CM transport.

If none is specified, the TIBRV default is used.

Type:	String
Required:	No
Mutability:	Restart required

**cm.sync.ledger**

Determines whether the CM transport synchronously writes to the ledger file.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	false
Legal values:	true   false

**cm.relay.agent**

Relay agent location for CM transport.

If none is specified, the TIBRV default is used.

Type:	String
Required:	No
Mutability:	Restart required

**cm.time.limit**

Time-to-live value for messages for CM transport.

If none is specified, the TIBRV default is used.

Type:	String
Required:	No
Mutability:	Restart required

**cm.listeners**

A space-separated list of listeners that will be registered to receive messages when they connect, for CM transport.

Type:	String
-------	--------

Required:	No
Mutability:	Restart required
<b>cm.discoveryInterval</b>	
The period on which a blank TibrvMsg will be sent out to enable any listeners to start the discovery process and receive messages, for CM transport.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>cmq.cm.name</b>	
CM name for CMQueue transport.	
If none is specified, the TIBRV default is used.	
Type:	String
Required:	No
Mutability:	Restart required
<b>cmq.worker.weight</b>	
Worker weight specification for CMQueue transport.	
If none is specified, the TIBRV default is used.	
Type:	String
Required:	No
Mutability:	Reactivation required
<b>cmq.worker.tasks</b>	
Worker tasks specification for CMQueue transport.	
If none is specified, the TIBRV default is used.	

Type:	String
Required:	No
Mutability:	Restart required
<b>cmq.scheduler.weight</b>	
Scheduler weight specification for CMQueue transport.	
If none is specified, the TIBRV default is used.	
Type:	String
Required:	No
Mutability:	Restart required
<b>cmq.scheduler.heartbeat</b>	
Scheduler heartbeat specification for CMQueue transport.	
If none is specified, the TIBRV default is used.	
Type:	String
Required:	No
Mutability:	Restart required
<b>cmq.scheduler.activation</b>	
Scheduler activation specification for CMQueue transport.	
If none is specified, the TIBRV default is used.	
Type:	String
Required:	No
Mutability:	Restart required
<b>cmq.complete.time</b>	
Scheduler complete time for CMQueue transport.	

If none is specified, the TIBRV default is used.

Type:	String
Required:	No
Mutability:	Restart required

### transformer

Class that transforms FIX Messages into TibrvMsg objects.

This class must implement [IMessageTransformer](#).

Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	com.camerontec.catalys.server.adapter.tibrv.TIBRVTransformer

### double.converter

Class to perform double to fixed-precision conversion.

This class must implement [IDoubleToDecimalConverter](#).

If this class is not specified, messages containing double values will result in an exception.

Type:	String
Required:	No
Mutability:	Restart required
Default value:	com.camerontec.catalys.util.data.DefaultDoubleToDecimalConverter

## TIBRVMessageSource

### id

ID of configuration element

Must be unique across the same configuration element types.

Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.adapter.tibrv.TIBRVConsumer
<b>subject</b>	
The subject on which to publish outbound messages or from which to read inbound messages.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>transport.type</b>	
Identifies the TIBRV transport type to use.	
Rendezvous offers 4 types:	
<ul style="list-style-type: none"> <li>• RVD (daemon)</li> <li>• RVA (agent)</li> <li>• CM (certified messaging)</li> <li>• CMQueue (distributed certified messaging)</li> </ul>	
Type:	String
Required:	No



Mutability:	Restart required
Default value:	RVD
Legal values:	RVD   RVA   CM   CMQ
<b>impl</b>	
The implementation type for TIBRV.	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	NATIVE
Legal values:	NATIVE   JAVA   SELECT
<b>reconnect.delay</b>	
Milliseconds to wait before reconnecting after a disconnection.	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	10000
<b>rvd.service</b>	
Service name for RVD transport.	
If none is specified, the TIBRV default is used.	
Type:	String
Required:	No
Mutability:	Restart required

**rvd.network**

Network name for RVD transport.

If none is specified, the TIBRV default is used.

Type:	String
Required:	No
Mutability:	Restart required

**rvd.daemon**

Daemon location for RVD transport.

If none is specified, the TIBRV default is used.

Type:	String
Required:	No
Mutability:	Restart required

**rva.host.name**

Host name for RVA transport.

If none is specified, the TIBRV default is used.

Type:	String
Required:	No
Mutability:	Restart required

**rva.port**

Port number for RVA transport.

If none is specified, the TIBRV default is used.

Type:	String
Required:	No

Mutability:	Restart required
<b>rva.tunnel.mode</b>	
Tunnel mode for RVA transport.	
If none is specified, the TIBRV default is used.	
Type:	String
Required:	No
Mutability:	Restart required
<b>cm.cm.name</b>	
CM name for CM transport.	
If none is specified, the TIBRV default is used.	
Type:	String
Required:	No
Mutability:	Restart required
<b>cm.request.old</b>	
Determines whether unacknowledged messages will be re-requested from senders for CM transport.	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	false
Legal values:	true   false
<b>cm.ledger.name</b>	
The CM ledger file for CM transport.	
If none is specified, the TIBRV default is used.	

Type:	String
Required:	No
Mutability:	Restart required
<b>cm.sync.ledger</b>	
Determines whether the CM transport synchronously writes to the ledger file.	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	false
Legal values:	true   false
<b>cm.relay.agent</b>	
Relay agent location for CM transport.	
If none is specified, the TIBRV default is used.	
Type:	String
Required:	No
Mutability:	Restart required
<b>cm.time.limit</b>	
Time-to-live value for messages for CM transport.	
If none is specified, the TIBRV default is used.	
Type:	String
Required:	No
Mutability:	Restart required
<b>cmq.cm.name</b>	

CM name for CMQueue transport.

If none is specified, the TIBRV default is used.

Type:	String
Required:	No
Mutability:	Restart required

#### **cmq.worker.weight**

Worker weight specification for CMQueue transport.

If none is specified, the TIBRV default is used.

Type:	String
Required:	No
Mutability:	Restart required

#### **cmq.worker.tasks**

Worker tasks specification for CMQueue transport.

If none is specified, the TIBRV default is used.

Type:	String
Required:	No
Mutability:	Restart required

#### **cmq.scheduler.weight**

Scheduler weight specification for CMQueue transport.

If none is specified, the TIBRV default is used.

Type:	String
Required:	No
Mutability:	Restart required

**cmq.scheduler.heartbeat**

Scheduler heartbeat specification for CMQueue transport.

If none is specified, the TIBRV default is used.

Type:	String
Required:	No
Mutability:	Restart required

**cmq.scheduler.activation**

Scheduler activation specification for CMQueue transport.

If none is specified, the TIBRV default is used.

Type:	String
Required:	No
Mutability:	Restart required

**cmq.complete.time**

Scheduler complete time for CMQueue transport.

If none is specified, the TIBRV default is used.

Type:	String
Required:	No
Mutability:	Restart required

**transformer**

Class that transforms TibrvMsg objects into FIX Messages.

This class must implement [IMessageTransformer](#).

Type:	String
Required:	No

Mutability:	Restart required
Default value:	com.camerontec.catalys.server.adapter.tibrv.TIBRVTransformer
<b>double.converter</b>	
<p>Class to perform double to fixed-precision conversion.</p> <p>This class must implement <a href="#">IDoubleToDecimalConverter</a>.</p> <p>If this class is not specified, messages containing double values will result in an exception.</p>	
Type:	String
Required:	No
Mutability:	Restart required
Default value:	com.camerontec.catalys.util.data.DefaultDoubleToDecimalConverter

## Task

<b>id</b>	
<p>ID of configuration element</p> <p>Must be unique across the same configuration element types.</p>	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
<p>Java class of this element's Runtime object.</p> <p>This class can be one of the standard CameronTec scheduling tasks or it can be a custom task, inheriting from <a href="#">TaskBase</a>.</p>	
Type:	String
Required:	Yes

Mutability:

Reactivation required

# ThreadStackLogger

## id

ID of configuration element

Must be unique across the same configuration element types.

Type:

String

Required:

Yes

Mutability:

Restart required

## class

Java class of this element's Runtime object.

Type:

String

Required:

No

Mutability:

Fixed to default value

Default value:

com.camerontec.catalys.server.test.ThreadStackLogger

## period

Number of milliseconds between each thread stack.

Type:

Integer

Required:

No

Mutability:

Reactivation required

Default value:

5000

Minimum value:

10

Maximum value:

60000



# Timerecord

id	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
class	
Java class of this element's Runtime object.	
Type:	String
Required:	Yes
Mutability:	Reactivation required
fromStart	
Controls whether the time record is started before processing or after.	
If true, recording starts before processing, and if false, it starts afterwards.	
Type:	Boolean
Required:	No
Mutability:	Reactivation required
Default value:	false
direction	
Controls whether the time is recorded for outbound or inbound messages	
Type:	String

Required:	No
Mutability:	Reactivation required
Legal values:	inbound   outbound
<b>defaultIntervalUpperBoundConfidence</b>	
The default confidence value (double) using which the histogram's upper bound interval is retrieved.	
Type:	String
Required:	No
Mutability:	Reactivation required
Default value:	0.95

## Timestamp

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>fromStart</b>	
Controls whether the time stamp is taken before processing or after.	
If true, the time stamp is taken before processing, and if false, it is taken afterwards.	
Type:	Boolean
Required:	No
Mutability:	Reactivation required

Default value:	true
<b>direction</b>	
Controls whether outbound or inbound messages are timestamped	
Type:	String
Required:	No
Mutability:	Reactivation required
Legal values:	inbound   outbound

## UNENCRYPTED

<b>encryptmethod</b>	
Encryption method used by this element.	
For this UNENCRYPTED element, this attribute has the fixed value of '0'.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	0

## ValidatorTemplateMessageSource

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes

Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.test.ValidatorTemplateMessageSource

## WebServerService

<b>id</b>	
ID of configuration element	
Must be unique across the same configuration element types.	
Type:	String
Required:	Yes
Mutability:	Restart required
<b>class</b>	
Java class of this element's Runtime object.	
Type:	String
Required:	No
Mutability:	Fixed to default value
Default value:	com.camerontec.catalys.server.web.WebServerService

# WriteQuorumSpecification

<b>writeQuorumAtLocalSite</b>	
The number of acknowledgements that must be received from the local site.	
Type:	Integer
Required:	Yes
Mutability:	Restart required
Minimum value:	0
<b>writeQuorumAtRemoteSites</b>	
The number of acknowledgements that must be received from the remote site.	
Type:	Integer
Required:	Yes
Mutability:	Restart required
Minimum value:	0
<b>writeQuorumAtAllSites</b>	
The number of acknowledgements that can be received from any site.	
Type:	Integer
Required:	Yes
Mutability:	Restart required
Minimum value:	0

# Appendix B. CatalysServer Startup Arguments

The following table lists the available arguments when starting the [CatalysServer](#) class.

Parameter	Description
-xmlconfig <config-file>	<p>The path to the configuration file.</p> <p>Required.</p>
-clusterNode <cluster-node-id>	<p>Identifies which node in a High Availability cluster to start. It must match the <code>id</code> of one of the <code>ClusterNode</code> definitions in the configuration.</p> <p>Optional.</p>
-resetseq	<p>Reset sequence numbers for all FIX sessions. This parameter can not be used with High Availability.</p> <p>Optional.</p>
-lock	<p>Put all FIX sessions in the locked state.</p> <p>A locked state means that no incoming connection attempts will be accepted and no outbound connection attempts will be made. This state is the same as blocked except it's not persisted and only valid during the life of that FIX server instance. The lock can be released via the <code>s_unlock</code> command.</p> <p>Optional.</p>
-skipXmlValidation	<p>Do not perform syntactic validation of the XML configuration against the DTD file.</p> <p>Semantic validation of the configuration is still performed by the Node once the XML configuration has been loaded. Specifying this option allows the server to be configured with elements and attributes which have not been specified in the DTD, which may be applicable for custom components.</p> <p>Optional.</p>

## CatalysServer Startup Arguments

Parameter	Description
-doNotBackupConfiguration	<p>Do not backup the configuration file when configuration changes are made at runtime.</p> <p>By default the configuration is saved in XML format to a timestamped file every time the runtime configuration is changed.</p> <p>Optional.</p>
-version	<p>Print the Node version information to stdout and then exit. When specified, all other arguments are ignored.</p> <p>Optional.</p>

# Appendix C. FIX Session Dialects

## Table of Contents

C.1. Introduction .....	701
C.2. Configuring a Dialect .....	701
C.3. Available Dialects .....	702
C.4. Implementations Requiring Dialects .....	707

## C.1. Introduction

Unfortunately, not all FIX implementations conform fully to the FIX standard. It is hoped that such implementations will eventually move into line with the standard. However, in the meantime, it may be necessary to communicate with them in their non-standard form. The problem is how to safely support non-standard communication without compromising our own strict compliance with the standard and without weakening the FIX standard itself.

The Node handles this general problem through the use of *dialects*. Each dialect is a well-defined departure from the standard low-level FIX protocol. Individual FIX sessions can be configured to support one or more dialects.

Dialects do not cover high-level application message protocol differences (such as correct handling of ClOrdID and OrigClOrdID). Although such departures from the standard are serious, they are usually best handled by the user's application. By contrast, if the low-level protocol is not functioning correctly, sessions can lock up completely or, in the worst case, messages can be completely lost without the user's application even being aware that there has been a problem.

## C.2. Configuring a Dialect

Dialects are configured as attributes of the `Session` element. The name of the dialect corresponds to the name of the attribute. Most dialect attributes are boolean and accept either `true` or `false`, while others accept string or integer values. For example:

```
<Session counterpartycompid="BROKER"
  compid="MARKET"
  fixversion="4.4"
  dialectAllowResendTimeout="true"
  dialectInsertCustomFieldsInHeader="5004=1,5006=0">
  <Connections>
    <SocketConnection id="conn" listenport="2002"/>
  </Connections>
```



```

<SessionManager>
  <SourceMessageProcessors/>
  <ListenerMessageProcessors/>
</SessionManager>
</Session>

```

## C.3. Available Dialects

- **dialectAllowLogonResponseWithoutEncryptMethod**

This dialect allows you to logon to a counterparty which does not specify the EncryptMethod (tag 98) in its Logon response message.

The FIX specification requires that the EncryptMethod tag be set both in the Logon request and response. However, some FIX engines do not process this tag and do not set it in the Logon response. With this dialect enabled the Node will accept Logon response messages which do not contain the EncryptMethod tag. However, the Catalys Node will still set the EncryptMethod tag in its Logon request.

This dialect only applies when the Node is the initiator of the FIX connection. That is, it does not modify the requirements for Logon messages accepted by the Node.

- **dialectAllowNoLengthOnRawData**

This dialect will attempt to process RawData field (tag 96) which do not have a corresponding RawDataLength field (tag 95).

The FIX specification requires that RawData field is immediately preceded by a RawDataLength field. This is necessary because the RawData field can contain binary data and that binary data may contain the standard FIX field delimiter, SOH. This means that this delimiter cannot be used reliably to terminate a raw data field.

However, some FIX implementations assume that the RawData field will not contain binary data and therefore omit the RawDataLength field. The Node will normally reject any message containing the RawData field without a RawDataLength field. If this dialect is set, however, the Node will simply use SOH (FIX field delimiter) as an indication of the end of RawData field.

- **dialectAllowResendTimeout**

This dialect allows you to set the [resendTimeout](#) attribute of the `Session` to trigger a timer whenever a resend request is sent. If this timer triggers before the requested messages start to arrive from the FIX counter-party, then processing will skip to the next available message that has been received.

For example, the Node receives message 20 when expecting 15, it sends a ResendRequest for 15 to 19. If `resendTimeout` is set to 10000ms and message 15 does not arrive within 10 seconds, a warning is logged and processing continues with message 20.



## Warning

Using this dialect will cause messages to be lost when the FIX counter-party does not respond to ResendRequests within the `resendTimeout` window. It is designed to be used with non-FIX compliant applications that cannot be changed to conform to the FIX specification.

- **dialectDisconnectAfterResendTimeout**

When used in conjunction with `dialectAllowResendTimeout` and `resendTimeout`, this dialect causes the session to logout and disconnect when there is no response to a resend request during the timeout period.

- **dialectAlwaysSendPossDupFlagAndOrigSendingTime**

This dialect adds the PossDupFlag and OrigSendingTime to each outbound FIX message odd if the message is not for resend. If the message is for resend, the values of PossDupFlag and OrigSendingTime will follow standard FIX specification. If the message is not for resend, PossDupFlag will have value "N" and OrigSendingTime is set to be equal to the value of SendingTime.

- **dialectAlwaysSendResendRequest**

This dialect causes the Node to generate a ResendRequest message every time a message is received with a sequence number higher than expected. Normally, if a first message is received with a sequence number higher than expected and other messages follow sequentially following the first, only one ResendRequest would be issued. With the dialect set, one will be issued for the first message and each subsequent one.

- **dialectDoNotSendTargetCompID**

Causes TargetCompID (tag 56) to be removed from every outbound FIX message.

- **dialectEnableShortTestReqIDFormat**

Causes TestReqID (tag 112) to be formatted as "HHmmss", where HH is hours 0-24, mm is minutes and ss is seconds.

- **dialectIgnoreIDFieldsInLogonResponse**

This dialect will not use the ID fields (compID, subID and locationID) that are returned in a Logon response message.

When a party sends a logon request, the TargetCompID, SubID and LocationID fields together define the party with whom we wish to start a FIX session. The other party must reply with a responding logon

message. Normally, the SenderCompID, SubID and LocationID in that response will be the same as the Target ID values in the request. If, however, they are not the same, the Node assumes that the other party wishes to use these new IDs and will use those IDs for referring to the party for the rest of the session.

However, some FIX implementations do not correctly set the Sender IDs in a logon response. If this dialect is set, the Node will use the original party Target IDs regardless of what IDs the party responds with.

- **dialectIgnoreInputMessageSequencing**

This dialect causes the Node to ignore out-of-sequence sequence numbers on incoming messages and process them immediately as if their sequence numbers were correct. If this dialect is set, all input messages are processed as if their sequence numbers were correct. Note the session's input sequence number will not be updated until logout.

- **dialectInitializeCipherOncePerSession**

This dialect relates to encryption. If set, the encryption cipher is initialized only once at the beginning of each session (at logon).

Normally the encryption cipher should be reinitialized for each FIX message. Otherwise, any missed or corrupted messages will result in the sending party's cipher and receiving party's cipher getting "out of sync".

- **dialectInsertCustomFieldsInHeader**

This dialect allows you to send custom fields in the header of every FIX message to a counterparty who is expecting these fields in the header. This parameter contains a comma-separated list of custom tag and value pairs.

For example: `dialectInsertCustomFieldsInHeader="5004=1,5006=0"`

- **dialectNumericSendingTime**

This dialect causes the SendingTime and OrigSendingTime fields to be numeric values instead of FIX-format date strings. The time is represented as the number of milliseconds since midnight, January 1, 1970 UTC. This can be useful for FAST, if the SendingTime and OrigSendingTime fields are defined as integers by the FAST template.

- **dialectOrigSendingTimeInGapFill**

This dialect adds the OrigSendingTime field to gap fill messages. The field will contain the current time.

We believe that this field has no meaning on a GapFill message and therefore do not normally include it. However, the Fujitsu FIX engine requires its presence. Fujitsu maintain that OrigSendingTime should be present on every message that has PossDup flag set. We do not believe that the FIX specification requires this. Moreover, we do not believe that this field can carry any useful information on a gap fill message and therefore it is completely redundant.

- **dialectProcessSkippedMessages**

This dialect requires that messages skipped as a result of a Sequence Reset or Gap Fill should be processed if they have been received by our application.

For example, suppose that we have received messages 1, 2, 4 and 5. We have missed message 3 so we will send a resend request. In the meantime we will have held up the processing of messages 4 and 5 until 3 has arrived. If the other side replies with a sequence reset or gap fill setting the new sequence number to 6, we would normally forget about message 3 and throw away messages 4 and 5. The sequence reset from the other side is telling us that it is okay to do this. This is standard FIX behaviour.

However, some FIX implementations assume that we will process *any message we have received*, even if they have sent us a sequence reset. This dialect instructs the FIX engine to behave in this non-standard fashion.

- **dialectResendRequestEndSeqNoAlwaysInfinity**

Some FIX implementations require that any ResendRequest message you send them must have the infinity value ("999999" for FIX 4.0 and 4.1, "0" for later versions) in the EndSeqNo field. This dialect causes the Node to generate ResendRequest messages with this field value. This is useful for CME iLink 2.0 and SWIFTNet FIX.

- **dialectSendNextExpectedMsgSeqNumOnLogin**

The FIX specification allows for message recovery at login using the NextExpectedMsgSeqNum tag, the use of which is optional, and used by agreement between parties. As the session acceptor, the Node will use this tag if the initial logon message from the session initiator contains it - so this dialect is not needed in this case. As session initiator, the Node will use this tag only if this dialect is enabled.

- **dialectSequenceResetToLastSeqno**

The FIX specification states that the NewSeqNo field in a Sequence Reset message should specify the "*next* sequence number immediately following the messages and/or sequence numbers being skipped".

However, some FIX engines place the *last* sequence number in this field. This dialect causes the Node to interpret this field as the *last* sequence number received. The next sequence number will be NewSeqNo + 1.

- **dialectIgnoreEmptyTags**

The FIX specification states that "All tags must have a value specified". By default, the Node enforces this restriction on inbound messages and will reject messages that have tags with no values.

If this dialect is enabled, this restriction will not be enforced and tags without values will be permitted.

- **dialectIgnoreBeginString**

In FIX, the BeginString field specifies the FIX version in use. If the Node receives a message where this does not match the session's configured version then the message will be rejected.

This dialect can be used to disable the check.

- **dialectIgnoreTimestampFormat**

In the FIX protocol, the SendingTime and OrigSendingTime fields must comply with a certain timestamp format. This format is actually FIX version dependent. The Node validates that these fields on incoming messages comply with the correct timestamp format and rejects messages that don't.

If this dialect is enabled, the Node will not validate the format of these fields on incoming messages.

- **dialectIgnoreCompanyID**

By default, the Node checks the consistency of the SenderCompID and TargetCompID fields over the course of a FIX session.

If this dialect is enabled, the Node will not validate these fields. Note that it is still possible for the Node to identify the session a message (except Logons) is intended for by looking at which socket the message was received on.

- **dialectIgnoreTagOrdering**

The Node enforces a certain ordering on the fields of incoming messages:

- BeginString must be first
- BodyLength must be second
- MsgType must be third
- CheckSum must be last

This dialect can be used to disable validation of the ordering of the MsgType field on incoming messages. Per the FIX spec, the order of these fields is fixed, but in order to support non-compliant engines sometimes you might need to allow an out-of-order MsgType field.

- **dialectAllowNonResetLogonMessagesMidSession**

Normally the Node does not permit Logon messages to be sent to an already logged on session unless the ResetSeqNumFlag (field 141) is set to 'Y'. If this dialect is enabled the Node accepts any Logon message and will reply with a Logon reply.

- **dialectHandleMissingPossDupOnGapFill**

The Node expects SequenceReset/GapFill messages to have PossDupFlag set to 'Y'. The Node will terminate the session when PossDupFlag is not set in the inbound message and its sequence number is lower than expected. If this dialect is enabled the Node will instead process the message as if it was received in sequence..

- **dialectDontSendPossDupFlag**

By default, the Node sets PossDupFlag to 'Y' in all messages that were sent before. If this dialect is enabled the PossDupFlag will not be set.

- **dialectToFixResendRequestMaxGap**

If the value of this attribute is greater than zero, the Node limits message gap range in ResendRequest to this value. This is required by CME drop-copy session. From the CME DropCopyImpact document: The maximum limit of messages allowed per resend request is 2500. In this scenario, multiple Resend Request messages must be submitted by the client spanning the desired sequence range.

- **dialectCustomResendRequestMaxGap**

If set to `true` and [dialectToFixResendRequestMaxGap](#) is greater than zero, the Node will not send a ResendRequest if the detected gap is greater than `dialectToFixResendRequestMaxGap`. In this case application can implement custom limit gap handling.

## C.4. Implementations Requiring Dialects

The following are known FIX implementations that require dialects in order for the Node to reliably communicate with them.

### EMX

Requires:

- [dialectProcessSkippedMessages](#)

When the EMX hub was written, FIX was at version 4.1. Unfortunately, that version of the FIX specification contained wording that misleadingly stated that the new sequence number in a gap fill should be "*the next sequence number to be transmitted*". This loose wording was corrected in version 4.2 of the specification to read "*the next sequence number immediately following the messages and/or sequence numbers being skipped*".

Unfortunately, EMX went with the literal wording in version 4.1. Using this interpretation, there is no implication that skipped messages can be ignored and the safest procedure is to process all skipped messages. EMX is not FIX and does not claim to be despite the fact that it is strongly based on the FIX standard. We are not aware of any move to change its behaviour in this respect.

### Reuters

Requires:

- [dialectProcessSkippedMessages](#)

Reuters have written their own FIX engine which does not currently appear to support Sequence Reset gap fills at all. It does not seem that it is possible for them to, say, in the above example, just skip message 3. In this situation, the safest procedure is to process skipped messages. In fact, we are aware of conditions where a session with Reuters can lock up unless this dialect is in use.

There has been some correspondence with Reuters on this. However, we are unaware of any move on their part to resolve the issue.

## Fujitsu

Requires:

- [dialectOrigSendingTimeInGapFill](#)

## CME (Chicago Mercantile Exchange)

Requires:

- [dialectAllowNoLengthOnRawData](#)
- [dialectIgnoreIDFieldsInLogonResponse](#)
- [dialectIgnoreInputMessageSequencing](#) (for TCP replay)

CME's "iLink FIX 4.2" uses the raw data field for a logon password and omits the raw data length field.

In addition, it does not return correct IDs in a logon response. If you try communicating using the IDs supplied in the logon response, CME will reject your messages.

CME also adds a non standard field, LoginRouteID, which must be present in the header of all messages, otherwise the messages are rejected. In order to support this, you need to use a non-default message factory for creating all messages. The Node installation supplies the [CmeMessageFactory](#) for this purpose which can be associated with the session using the [setFromFIXMessageFactory](#) and [setToFIXMessageFactory](#) methods.

The special CME TCP replay market data FIX session requires [dialectIgnoreInputMessageSequencing](#) because replayed market data messages come back with their original sequence numbers. However, this dialect should not be used elsewhere on normal CME FIX sessions.

## CME iLink 2.0

Requires:

- [dialectAllowNoLengthOnRawData](#)
- [dialectAlwaysSendResendRequest](#)

- [dialectIgnoreIDFieldsInLogonResponse](#)
- [dialectOrigSendingTimeInGapFill](#)
- [dialectProcessSkippedMessages](#)
- [dialectResendRequestEndSeqNoAlwaysInfinity](#)

The [CMEiLink2MessageFactory](#) should also be used.

## CSFB (Credit Suisse First Boston)

Requires:

- [dialectInitializeCipherOncePerSession](#)

This is only required if you are using FIX encryption with CSFB and you are communicating with their in-house FIX engine.

## Macgregor Predator OMS

Requires:

- [dialectSequenceResetToLastSeqno](#)

Without the use of this dialect, the Node will see the next incoming message as 1 too large and request a resend of the gap. On receipt of another Sequence Reset, the Node breaks the connection because the sequence number is being set back 1, which is illegal in FIX.

Macgregor is aware of this deviation from FIX, but has made no move to correct it.

## SFE (Sydney Futures Exchange)

Requires:

- [dialectAllowLogonResponseWithoutEncryptMethod](#)
- [dialectAlwaysSendPossDupFlagAndOrigSendingTime](#)
- [dialectDoNotSendTargetCompID](#)
- [dialectEnableShortTestReqIDFormat](#)
- [dialectIgnoreIDFieldsInLogonResponse](#)
- [dialectInsertCustomFieldsInHeader](#)

The SFE FIX engine accepts Logon requests containing the EncryptMethod tag, however, it does not set it in its Logon response. Normally the Node would reject such a Logon response, however, it will be accepted when `dialectAllowLogonResponseWithoutEncryptMethod` is enabled.



## FIX Session Dialects

The SFE FIX engine expects PossDupFlag(43) and OrigSendingTime(122) set in every incoming FIX message even if the message is not for reset. If the message is not for resend, PossDupFlag is set with value "N" and OrigSendingTime is set to be equal to SendingTime.

The SFE FIX engine also expects custom tag 5006 in the header portion of every incoming FIX message. When sending FIX message to SFE using the Node, this custom tag can be added by setting `dialectInsertCustomFieldsInHeader` to 5006=0.

According to the FIX specification provided by SFE, it does not expect incoming message to have TargetCompID (field 56) set and does not send TargetCompID in response FIX message to connected party. Additionally, SFE FIX engine expects TestReqID (field 112) in incoming FIX test request message in the format of "HHmmss".

# Appendix D. Persistent Collections Properties

## Table of Contents

D.1. Introduction .....	711
D.2. Properties .....	711
D.2.1. Common Collection Properties .....	711
D.2.2. Specific Collection Properties .....	715
D.3. Compacting Collections .....	720

## D.1. Introduction

Any Node component with a requirement for persistence (whether this is session state persistence or persistence of some other type of data) uses the the general Persistent Collections framework to generate and process its files.

Collections properties are used for components which come supplied with the Node, such as the [JournalingPersister](#) and also for custom components which make use of persistence (see [Persistent Objects](#)). These properties are passed in to the component via a child `Properties` element.

The name *collection* is used in the sense of a Java collection: a group of objects, in this case a group of objects written to and read from disk. These objects can be messages, as in the case of the `JournalingPersister`'s output FIX messages; or they can be an `Integer`, as in the case of the `JournalingPersister`'s input sequence number. The number of the group can be very large, as in the case of messages or can be limited to 1, as in the case of a sequence number.

There are properties common to all collections. Depending on the type and volume of data, different collection types (indexed list, high capacity list and mapped data) are available. These also have specific properties.

## D.2. Properties

### D.2.1. Common Collection Properties

Property	Definition
directoryName	The location of the persistence file associated with the collection.  Required.

Property	Definition
name	<p>The name under which a collection is registered in the collection registry. If this property is not specified, the value of <code>directoryName</code> is used instead.</p> <p>If using a collection in a High Availability configuration, it is recommended to specify an explicit collection name through this property when the collection's directory name is long (for example if it is an absolute path). This is because collection names are exchanged between cluster nodes for the purpose of data replication and synchronization.</p> <p>Names specified using this property have to be unique across all collections bound to the collections registry.</p> <p>Optional.</p>
isHighCapacityList	<p>Whether this is a high capacity list.</p> <p>Standard capacity lists provide excellent read and write performance and contain about 12 bytes of overhead for each element. Initialization time can be affected when the list grows large since, in order to maintain excellent write performance there are no disk indices, which means they must be deduced on initialization. Standard capacity lists also have the additional specific properties specified in <a href="#">Standard Capacity List Properties</a>.</p> <p>High capacity lists preserve low start times. They have slightly worse read and write performance than standard capacity lists and they have fixed, configurable memory overhead (the default is 1MB). High capacity lists also have the additional specific properties specified in <a href="#">High Capacity List Properties</a>. In addition to the journal files written for high capacity collections, there is also an index file in the collection directory.</p> <p>Optional. Defaults to <code>false</code>.</p>
synchAfterWrite	<p>Whether to synchronize with the file after every write.</p> <p>In HA configurations, if this property is set to <code>true</code> and there is a failed replication attempt, then the outstanding write operations are synchronized with the underlying file.</p>

Property	Definition
	<p>In standalone configurations, each successful write is synchronized with the underlying file. This can incur a significant performance overhead.</p> <p>Optional. Defaults to <code>false</code>.</p>
<code>useWriteBuffering</code>	<p>Whether to use write buffering.</p> <p>If this property is set to <code>true</code> then data is only flushed to disk when there is <code>writeBufferSize</code> data to write, or the <code>writeBufferFlushingDelay</code> is exceeded. Setting the <code>writeBufferSize</code> to the disk block size increases performance but adds risk that data will be lost on an application restart.</p> <p>Optional. Defaults to <code>false</code>.</p>
<code>writeBufferSize</code>	<p>The size, in bytes, of the write buffer to use if write buffering is turned on.</p> <p>Note: When a write request larger than this buffer size is encountered, the write buffer is extended so that it can accommodate this write request. However, this is a costly operation, therefore it is recommended to use a sensible size from the beginning rather than relying on this buffer extension mechanism.</p> <p>Optional. Defaults to 64kB.</p>
<code>writeBufferFlushingDelay</code>	<p>The maximum amount of time, in milliseconds, that data should remain in the writing buffer if write buffering is turned on.</p> <p>Note: Since this implementation does not use a flushing thread, there is no guarantee that a flush will happen at regular time interval, since writes may be infrequent. Therefore, this delay really comes into play when writes are frequent but the writing buffer takes a long time to fill up.</p> <p>Optional. Defaults to 1s. Can be between 0.5s and 5s.</p>
<code>randomAccessReadBufferSize</code>	<p>The size, in bytes, of the read buffer to use when performing random access reads.</p> <p>This is an advanced property which should be changed with care. The default should be suitable in a majority of situations.</p>

Property	Definition
	<p>In particular, using large values can cause serious performance degradation.</p> <p>Optional. Defaults to 128 KB. Value must be between 1 KB and 128 MB.</p>
scanningReadBufferSize	<p>The size, in bytes, of the read buffer to use when scanning the underlying journal, either at start-up, or after compacting.</p> <p>This property has a relatively low default value in order to avoid consuming too much heap memory for collections that are not likely to take a long time to be scanned. For very large collections though, using a large value is likely to have noticeable effects on scanning performance, since fewer I/O operations will be required to go through the entire journal.</p> <p>Optional. Defaults to 128 KB. Value must be between 1 KB and 128 MB.</p>
rollJournalFileEvery	<p>The maximum size, in bytes, that the current journal file is allowed to reach.</p> <p>Once the current journal file reaches that size, a new journal file is created and subsequent writes are made to this new file. Rolling always happens before the limit is reached, which means that journal files can never exceed the limit. A value less than zero disables this rolling functionality, meaning that the journal file will keep growing as long as the file-system allows.</p> <p>Optional. Defaults to 2GB.</p>
recordVersionHistory	<p>Whether or not the store's indexer should keep an in-memory record of the offsets of the last write requests in the journal in order to be able to access version history.</p> <p>When turned on, the actual size of the history is given by the value of property <code>versionHistorySize</code>. This feature is normally only needed when running in High Availability configurations, where it is necessary to be able to replay version history for synchronization purposes. The memory footprint when this feature is turned on is 8 bytes times the size of the history. So for a default history size of 10,000, the footprint is 80 kb.</p>

Property	Definition
	Optional. Defaults to <code>false</code> .
<code>versionHistorySize</code>	<p>The size of the in-memory record of write request offsets maintained by the store's indexer when version history recording is turned on.</p> <p>Using a smaller value reduces the memory footprint and start-up time (because it reduces the area in the journal file that must be scanned to reconstruct the history), but increases the likelihood that the entire journal file must be scanned to service a version history request because the requested version is outside the record. Using a larger value increases the memory footprint and start-up time but reduces the likelihood that the entire journal file must be scanned when searching for a version.</p> <p>Optional. Defaults to 10,000. Can be between 1,000 and 1,000,000.</p>
<code>filePathLevelsInLogger</code>	<p>Specifies the number of path components of this store's backing directory to include in the name of the logger used to print log messages.</p> <p>Optional. Defaults to the base name of the directory, which corresponds to a depth of one. Value can be between 1 and the maximum file path levels.</p>
<code>replicated</code>	<p>Specifies whether the contents of this collection should be replicated across the cluster when used in a high availability context.</p> <p>Optional. Defaults to <code>true</code>, which means that collections are replicated by default. Ignored when not used in a high availability context.</p>

## D.2.2. Specific Collection Properties

### D.2.2.1. Standard Capacity Collection Properties

These specific properties only apply when the [isHighCapacityList](#) property is set to `false`.

Property	Definition
<code>initialIndexCapacity</code>	If many entries are to be stored in the store, creating it with a sufficiently large capacity will allow the entries to be stored more

Property	Definition
	<p>efficiently than letting it perform automatic array copies as needed to grow the in-memory index.</p> <p>Optional. Defaults to 10,000. Value can be between 1 and <math>2^{31} - 1</math>.</p>

### D.2.2.2. High Capacity List (Indexed) Collection Properties

These specific properties only apply when the [isHighCapacityList](#) property is set to `true`.

Property	Definition
<code>indexerMemoryLimit</code>	<p>The limit, in bytes, to the amount of memory that this store's indexer is allowed to use for its index backlog.</p> <p>The backlog contains the index entries that have yet to be written to disk. In most cases (i.e. when data in write requests is not too large), it takes significantly less time to append a write request to the end of a journal file than to add an index entry to the on-disk index. Therefore, under a constant load of writes, the indexer's backlog will grow until it reaches this limit. Once the backlog is full, further writes will be throttled to the speed of the index persister thread. Using a larger limit can be useful if it is necessary to absorb data bursts without performance degradation for a longer time.</p> <p>Optional. Defaults to 1MB. Value can be between 0.5MB and as much as the heap size allows.</p>
<code>reportIndexerThrottling</code>	<p>Whether the store's indexer should regularly report the number of writes throttled and the average extra latency due to throttling.</p> <p>When enabled, a warning will be printed to the logs every N write operations to the store (N being the value given by property <code>reportIndexerThrottlingEvery</code> below) if at least one operation was throttled.</p> <p>Optional. Defaults to <code>true</code>.</p>
<code>reportIndexerThrottlingEvery</code>	<p>The occurrence with which throttled writes should be reported as a warning in the logs by the store's indexer (provided there is at least one throttled write to report) if throttle reporting is turned on (using property <code>reportIndexerThrottling</code> above).</p> <p>Optional. Defaults to 1,000,000. Value can be between 1,000 and 1,000,000.</p>

Property	Definition
<code>isBytesCachingEnabled</code>	<p>Whether the store's underlying journal should make use of a cache to increase reading performance.</p> <p>Refer to the description of Map property <a href="#">isBytesCachingEnabled</a> for details.</p>
<code>bytesCacheCapacity</code>	<p>The capacity, expressed as a number of bytes, of the journal's bytes cache.</p> <p>Refer to the description of Map property <a href="#">bytesCacheCapacity</a> for details.</p>
<code>bytesCacheAverage-PayloadSize</code>	<p>The expected average size, expressed as a number of bytes, of entries stored in the journal's bytes cache.</p> <p>Refer to the description of Map property <a href="#">bytesCacheAveragePayloadSize</a> for details.</p>
<code>bytesCacheObjectName</code>	<p>The JMX object name under which the cache should register an MXBean exposing its sizing and utilisation data at runtime. Refer to the MXBean's self-exposed attribute descriptions for details about these attributes.</p> <p>Refer to the description of Map property <a href="#">bytesCacheObjectName</a> for details.</p>

### D.2.2.3. Map Collection Properties

These properties only apply to `Map` collections.

Property	Definition
<code>isCachingEnabled</code>	<p>With caching enabled the underlying implementation maintains an in memory cache for the purpose of improving read performance. Even though the read performance of the underlying persistent store is likely to be very good, there is still a cost associated with deserializing data. This cost may be avoided altogether in many cases via an adequate caching policy.</p> <p>Note that this property is only applicable to map collections.</p> <p>Optional. Defaults to <code>true</code>.</p>



Property	Definition
maxCacheElements	<p>The maximum number of elements in the memory cache. Only used if <code>isCachingEnabled</code> is set to <code>true</code>.</p> <p>Optional. Defaults to 100.</p>
indexLoadFactor	<p>The load factor of the in-memory hash map that represents the store's index.</p> <p>Optional. Defaults to 0.5. Valid values are <math>&gt; 0</math> and <math>\leq 1.0</math>.</p>
isBytesCachingEnabled	<p>Whether the store's underlying journal should make use of a cache to increase reading performance.</p> <p>When enabled, the store's underlying journal caches bytes in memory to avoid having to read bytes from disk too often. This is a different type of cache to the cache that gets created when using property <code>isCachingEnabled</code>, which caches objects representing map values on the heap and is therefore somewhat limited in size. A journal's bytes cache, by contrast, only caches bytes, and makes use of off-heap memory to do so, which means that it is only limited by the amount of physical memory available on the local machine. This type of cache is particularly well suited for collections storing large to very large values and from which reads are very frequent. In such cases, if sized properly, the cache can potentially eliminate the need for disk I/O.</p> <p>Optional. Defaults to <code>false</code>, meaning that the journal does not cache bytes by default.</p> <p>A journal's bytes cache combines a small synchronous on-heap L1 cache with a very large asynchronous off-heap L2 cache. It therefore requires a dedicated thread to exchange data with its L2 cache. The total capacity of the bytes cache is expressed as a number of bytes (for example, 1 GB), and the amount of heap memory used by the L1 cache never exceeds 5% of that capacity.</p> <p>The bytes cache uses an LRU (Least Recently Used) policy to evict old entries once at capacity.</p> <p>In order to get the best out of the amount of off-heap memory that it can use, the L2 cache needs to be calibrated, which is a one-off operation which consists in segmenting the off-heap memory into small data chunks used to store actual payload.</p>

Property	Definition
	<p>The quantity and size of those chunks depends on the expected size of the payload, which therefore needs to be configured. This is expressed as the cache's average payload size, which is the average size in bytes (for example, 1 KB) of the <i>values</i> stored into the persistent collection. This average payload size does not need to be (and sometimes cannot be) too precise, but it is still important to configure a size that is close enough to the actual payload size:</p> <ul style="list-style-type: none"> <li>• If the actual payload size is significantly smaller than the configured size, the cache will deliver optimal performance but will not be using all of the off-heap memory that is allocated to it.</li> <li>• If the actual payload size is significantly larger than the configured size, the cache will make use of all its off-heap memory but its reading performance will be slightly deteriorated.</li> </ul> <p>In order to ease the configuration of the cache's average payload size, it is possible to get the cache to register a JMX MBean that exposes various sizing and utilisation attributes that can be used to detect a suboptimal configuration.</p> <p>The calibration of the journal's bytes cache can take up to a few seconds when done for the first time. For that purpose, the cache persists its calibration data into a directory called <code>calibration</code> located within the persistent collection's directory. This calibration data is reused when present. If the calibration directory gets deleted, then a new calibration will take place the next time the collection is loaded.</p>
bytesCacheCapacity	<p>The capacity, expressed as a number of bytes, of the journal's bytes cache.</p> <p>This is the cache's total capacity, which is split between an on-heap L1 cache which takes up to 5% of the capacity, and an off-heap L2 cache. Once at capacity, the cache will evict its oldest entries using an LRU (Least Recently Used) policy.</p> <p>Mandatory when <code>isBytesCachingEnabled</code> is true, ignored otherwise. Value must be at least 128 MB and cannot exceed the amount of physical memory available on the local machine.  <i>Warning:</i> Specifying a capacity that is larger than the amount of available physical memory will cause the JVM to crash.</p>

Property	Definition
bytesCacheAverage-PayloadSize	<p>The expected average size, expressed as a number of bytes, of entries stored in the journal's bytes cache.</p> <p>This size corresponds to the size of <i>values</i> stored in the persistent collection. In the case of a persisted map, the key size does not count, the cache is only used to store value bytes.</p> <p>Mandatory when <code>isBytesCachingEnabled</code> is true, ignored otherwise. Value must be at least 64 bytes and should ideally be as close as possible to the actual average payload size. Significant differences between the configured value and the actual average payload size will not cause the cache to malfunction, but it may not make the best out of its off-heap memory (in cases where the actual payload size is much larger than the configured value) or it won't deliver optimal reading performance (in cases where the actual payload size is much smaller than the configured value). It is therefore recommended to use the JMX MXBean that the cache can optionally expose to determine whether this value is properly configured.</p>
bytesCacheObjectName	<p>The JMX object name under which the cache should register an MXBean exposing its sizing and utilisation data at runtime. Refer to the MXBean's self-exposed attribute descriptions for details about these attributes.</p> <p>This MXBean can be used to determine whether the cache was configured properly and therefore makes the best out of its off-heap memory and delivers optimal performance.</p> <p>Optional when <code>isBytesCachingEnabled</code> is true, ignored otherwise. If omitted, no MXBean gets registered. If present, value must be a valid JMX object name and must not conflict with the name of another JMX MXBean.</p>

## D.3. Compacting Collections

To ensure that collections do not grow without bound, standard Node components, which use persistent collections, clear and compact their files as part of their end-of-day procedure. This procedure can be invoked by calling the [s\\_end\\_of\\_day](#) CLI command or [scheduling](#) the [EndOfDayTask](#).

Support is also provided to compact persistent collections at other times. This is accomplished by scheduling the [CompactTask](#).

## Persistent Collections Properties

This task is configured by providing the [name\(s\)](#) of the collection(s) to compact, as a property. For example:

```
<Task id="compact" class="com.camerontec.catalys.server.task.CompactTask">
  <Properties id="props">
    <Property name="names" value="logs/dir1:logs/dir2"/>
  </Properties>
</Task>
```



### Note

While compacting a collection is thread-safe, it can be a time consuming and resource intensive process and should be performed only while a session is inactive.

# Appendix E. Error Codes

## Table of Contents

E.1. Errors .....	722
E.1.1. General Errors .....	722
E.1.2. Send Errors .....	722
E.1.3. Bad Data Errors .....	722
E.1.4. Fatal Protocol Errors .....	723
E.1.5. Non Fatal Protocol Errors .....	725
E.1.6. Logon Errors .....	725
E.1.7. Message Errors .....	726
E.1.8. Routing Errors .....	728
E.1.9. Timeout Errors .....	728
E.1.10. SCP Errors .....	728

## E.1. Errors

### E.1.1. General Errors

#### Error 0: Any

This is a general error. See the message printed with the error for the specific problem.

### E.1.2. Send Errors

#### Error 1: Send Failure

This error occurs when a message cannot be sent.

#### Error 2: Message List Send Error

This error occurs when a list of messages is being sent, and the sending of any of them fails.

### E.1.3. Bad Data Errors

#### Error 101: Bad Encoded Length Type

This error occurs during message creation, when the encoded length is not found in the field infos structure.

## **Error 109: Unknown Encoding**

This error occurs during message creation, when an encoded field is added to message and its value is null, its encoding is null, or its encoding is invalid.

## **Error 110: Unknown FIX Version**

This error occurs during session logon, when the FIX version contained within the begin string field (FIX tag 8) has an incorrect prefix or is an invalid value. Valid values are: "2.7", "3.0", "4.0", "4.1", "4.2", "4.3", "4.4", "5.0", "5.0SP1", "5.0SP2".

## **Error 112: Unknown FIX Transport Version**

This error occurs during session login, when the FIX transport version contained within the begin string field (FIX tag 8) does not equal "1.1".

## **E.1.4. Fatal Protocol Errors**

### **Error 300: Application Rejected Party**

This error occurs during session logon, when the parties fields SenderCompID (FIX tag 49), SenderSubID (FIX tag 50), SenderLocationID (FIX tag 142), TargetCompID (FIX tag 56), TargetSubID (FIX tag 57), TargetLocationID (FIX tag 143) in a received logon message do not match those configured for the session.

### **Error 301: Message Sequence Number Too Low**

This error occurs during message processing, when the received sequence number is lower than expected. This is a fatal error because there is no way to recover when the sequence number corresponds to a message that has already been processed.

### **Error 302: Reset to Lower Sequence Number**

This error occurs during the processing of a sequence reset message (FIX msg type 4), when the new sequence number (FIX tag 36) is lower than the value of the last sequence number processed.

### **Error 303: Session not Started**

This error occurs when a resend request (FIX msg type 2) is received but the session is not logged on, or if the session attempts to send a message and the session is not started.

### **Error 304: Session is Blocked**

This error occurs during session establishment, when there is an attempt to connect the session while it is blocked.

## **Error 305: Resend Request for Unsent Messages**

This error occurs when a resend request message (FIX msg type 2) is received, and it is requesting message sequence numbers (FIX tags 7 and 16) that have not already been sent.

## **Error 306: Out of Sequence Resend Request**

This error occurs when a resend request (FIX msg type 2) is received and it is out of sequence itself.

## **Error 307: Both Ends Strategy Activation Failure**

This error occurs during session connection. If both parties of the Session are Catalys Nodes then a set of capabilities can be negotiated between the nodes, and an optimal set (strategy) can be selected. This error occurs when the two ends of the session cannot successfully negotiate a strategy.

## **Error 308: Next Expected Sequence Number Too High**

This error occurs during session logon. A FIX logon message can optionally contain the next expected sequence number (FIX tag 789: NextExpectedSequenceNumber). When a FIX session party receives a logon message that does contain this field, it checks its value against its next outgoing message sequence number. If they are equal it proceeds and if the next expected sequence number is lower than the number the party was expecting to send, then it gap fills.

This error occurs if the next expected sequence number is higher than the one the party was expecting to send, as it cannot recover messages that it has no record of sending.

## **Error 309: Garbled Message Received**

This error occurs during message processing, when a message received by a FIX party cannot be decoded. This can happen because:

- The message does not have a correctly formed 'Begin String'
- The message does not have the 'Body Length' tag as the second field of the message
- The 'Body Length' of the message does correspond to the actual length of the message
- The calculated checksum of the message does not match the 'Checksum' value.

If two or more of these messages are received consecutively then the session is logged out.

## **Error 310: First Message must be Logon**

This error occurs if the first message received is not a logon message.

## E.1.5. Non Fatal Protocol Errors

### Error 400: Message Exception

This error occurs whenever there is a problem with an incoming message but the problem is such that system can continue processing.

The message given with this log message will identify the specific problem with the message. Examples include:

- The value of a field in an incoming message cannot be converted to the correct type e.g. if there is a floating point number where there should be an integer
- If the message is missing some mandatory data e.g.
  - the message is a reject and it does not contain the sequence number of the message that is being rejected (Tag 45: 'RefSeqNum')
  - A logon message over an encrypted session is missing encryption information
  - A logon message over an encrypted session has unrecognizable encryption information
- The session times out waiting for its next message

### Error 401: Missing Messages

This error occurs if a message with a greater than expected sequence number is received.

The log message for this error will include the messages that are missing, and a resend request (FIX Msg Type 2) for those messages will be sent to the other party.

### Error 402: Not Allowed in Strict Mode

Strict mode (strict adherence to the FIX protocol) is the default setting for the Catalys Node Server. This mode can be switched off in certain certification and testing scenarios.

This error occurs when if an application attempts to perform an operation which would break the adherence to the FIX protocol. e.g. switching off the internal processing of FIX messages.

## E.1.6. Logon Errors

### Error 500: General Logon Failure

This error occurs when the logon fails, and the reason for the failure is not one of the specific failures described below.



## Error 501: General Logon Rejection

This error occurs when instead of receiving a logon message in response to sending one, another message type is received.

If the received message contains a text field (FIX tag 58) then then this will be logged with the error. Otherwise the message type (FIX tag 35) of the received message will be logged.

## Error 502: Rejected Encryption Method

This error occurs when the encryption method (FIX tag 98) contained within the logon message does not match the method configured locally.

## Error 503: Rejected Session Security Information

This error occurs when the session security information contained in the raw data field (FIX tag 96) does not match the session security information configured locally.

## Error 504: Logon Received Mid Session

This error occurs when a logon message is received mid-session i.e. after a successful login had previously been negotiated, and the Session attribute `dialectAllowNonResetLogonMessagesMidSession` is set to false. (If this dialect is true the the mid-session logon is ignored).

## E.1.7. Message Errors

### Error 600: Bad Data Value

This error occurs when a message contains an illegal value in one of its fields. The tag and value is printed in the message that is logged with the error.

The error can happen if `ConditionValidFIX` condition is included in the message processor chain of an application (on inbound or outbound messages), or if any of the standard fields in a message are malformed.

### Error 610: Missing Encryption Session Information

This error occurs during logon processing, on an encrypted session, when a logon response is received and the Session Info structure is missing from the raw data field (FIX tag 96) See [1].

### Error 613: Missing Required Field

This error occurs during message validation, when the message is missing a required field. The field that is missing will be shown in the error message, and can include:

- MsgType (FIX tag 35)
- BeginSeqNo (FIX tag 7) of a resend request (FIX msg type 2)
- EndSeqNo (FIX tag 16) of a resend request (FIX msg type 2)

## Error 615: Unknown Party

This error occurs when an adapter component cannot send a message across a FIX session because the comp/subcomp/location fields of the message do not unambiguously specify a FIX session.

## Error 616: Unrecognizeable Encyption Session Information

This error occurs during logon processing, on an encrypted session, when a logon response is received and the Session Info structure contained in the raw data field (FIX tag 96) is the wrong length. See [1].

## Error 617: Inbound Validation Error

This error occurs during inbound message validation, when the validation fails. Reasons for this failure can include:

- BeginString (FIX tag 8) has an invalid value
- (For FIX 5.0, DefaultAppVer field (FIX tag 1137) is missing or has an invalid value
- A field is empty
- A field cannot be converted to the correct type
- BeginString (FIX tag 8) contained in message does not match that configured on the session
- MsgType (FIX tag 35) is invalid
- SenderCompID (FIX tag 49) does not match that configured on the session
- TargetCompID (FIX tag 56) does not match that configured on the session
- SendingTime (FIX tag 52) is not in UTC format
- PossDupFlag (FIX tag 43) does not contain 'Y' or 'N'
- PossResend (FIX tag 97) does not contain 'Y' or 'N'

## Error 619: Message Serialization Failed

This error occurs during outbound message serialization, when the serialization fails. Reasons for this failure can include:

- Message contains a field with a negative tag number.

## E.1.8. Routing Errors

### Error 700: No Listener for Connection Point

This error occurs during message processing, when a message processor tries to send a message to an end point which has no listeners, that is there is no destination which is listening for the message.

## E.1.9. Timeout Errors

### Error 800: Timed Out Waiting for Input

This error occurs when the server takes too long to read a message from its input processing queue. In response to this condition, it will send a TestRequest message (FIX msg type 1). If the response to this message also times out then a standard I/O exception will be reported.

### Error 801: Timed Out Waiting for Test Request Heart Beat

This error occurs when the server does not receive a HeartBeat message (FIX msg type 0) within the two heart beat intervals (FIX tag 108 in the Logon message (FIX msg type A)) after sending a Test Request (FIX msg type 1).

The server responds to this condition by sending a logout message and closing the connection.

### Error 802: Timeout Out Waiting for Other Messages

This error occurs when a response to a logon or logout message is not received within a heart beat interval (FIX tag 108 in the Logon message (FIX msg type A)).

## E.1.10. SCP Errors

### Error 900 and 901: SCP Alert Received/Sent

This error occurs on a FAST session, when an SCP (session control protocol) alert is received or sent. This message provides as much information as possible regarding the reason for termination of the session. The message printed with this error has the following information:

- the Severity of the alert
- the Code of the alert, one of
  - Close: Normal termination
  - Unauthorized: Peer not authorized
  - UnknownTemplateId: The decoder cannot find a template name associated with a template identifier. The template identifier is supplied in the Value field

- **UnknownTemplateName:** The decoder cannot find a template definition associated with a template name. The template identifier is supplied in the Value field if available. The name of the missing template is included in the Description field.
- a Value associated with the alert
- a Description of the alert

For more information, refer to [2].

### **Error 1000: Hello Message Expected**

This error occurs on a FAST session, when the first message received is not a Hello message.

### **Error 1001: Not Supported In Version 1.1 Level 1**

This error occurs when any SCP message aside from a Reset message is received, as this level of the protocol only supports the Reset message.

### **Error 1002: Not Supported in Version 1.1 Level 2**

This error occurs when any SCP message aside from a Reset, Hello or Alert is received as these are the only SCP messages supported in this level of the protocol.

### **Error 1003: Should only receive one Hello message**

This error occurs when multiple SCP Hello messages are received. The protocol specifies that only one should be received.

## **References**

[1] See the Key Exchange section of the FIX Protocol Document: PGP-DES-MD5 and PEM-DES-MD5 Overview (security.doc).

[2] See the Fast Session Control Protocol Specification (FAST Session Control Protocol 1.1.doc).

# Appendix F. JMX MBean Reference

## Table of Contents

F.1. Platform MBeans .....	730
F.2. Server Management MBeans .....	732
F.2.1. Viewing Basic Node Information .....	732
F.3. Session Management MBeans .....	734

## F.1. Platform MBeans

The management package introduced in Java 5 defines a number of MBeans called *platform MBeans*, or *MXBeans*. Each MXBean encapsulates a single functional area of the JVM, such as:

- The number of classes loaded
- Uptime of the JVM
- The amount of memory consumed
- The number of threads running
- Statistics about thread contention
- etc.

Beginning with Java 5, the JVM has included a built-in MBean server called the *platform MBean server*, in which MXBeans are registered.

The Catalys management infrastructure uses the JVM's platform MBean Server to register all Server MBeans as well. As a consequence, all JVM MXBeans are exposed to the LMA and MA.

In order to guarantee a consistent naming scheme across all managed applications, the MA's publisher module exposes JVM MXBeans under the following names:

Resource Managed	Object Name
Class loader	<code>com.camerontec:Host="&lt;hostname&gt;", type="Host.CatalysServer.JVM.ClassLoading", CatalysServer="&lt;serverName&gt;"</code>
Compilation	<code>com.camerontec:Host="&lt;hostname&gt;", type="Host.CatalysServer.JVM.Compilation",</code>

## JMX MBean Reference

Resource Managed	Object Name
	CatalysServer="<serverName>"
Garbage collectors	com.camerontec:name="<garbageCollectorName>" , Host="<hostname>" , type="Host.CatalysServer.JVM.GarbageCollector" , CatalysServer="<serverName>"
Diagnostics for the HotSpot VM	com.camerontec:Host="<hostname>" , type="Host.CatalysServer.JVM.HotSpotDiagnostic" , CatalysServer="<serverName>"
Logging	com.camerontec:Host="<hostname>" , type="Host.CatalysServer.JVM.Logging" , CatalysServer="<serverName>"
Memory	com.camerontec:Host="<hostname>" , type="Host.CatalysServer.JVM.Memory" , CatalysServer="<serverName>"
Memory managers	com.camerontec:name="<memoryManagerName>" , Host="<hostname>" , type="Host.CatalysServer.JVM.MemoryManager" , CatalysServer="<serverName>"
Memory pools	com.camerontec:name="<poolName>" , Host="<hostname>" , type="Host.CatalysServer.JVM.MemoryPool" , CatalysServer="<serverName>"
Operating System	com.camerontec:Host="<hostname>" , type="Host.CatalysServer.JVM.OperatingSystem" , CatalysServer="<serverName>"
JVM Runtime	com.camerontec:Host="<hostname>" , type="Host.CatalysServer.JVM.Runtime" , CatalysServer="<serverName>"
Threading	com.camerontec:Host="<hostname>" , type="Host.CatalysServer.JVM.Threading" , CatalysServer="<serverName>"

For further information on JVM MXBeans, please refer to [Java SE Monitoring and Management](#), which provides, amongst other things, pointers to the detailed API specifications of the above MXBeans.

## Registering and Using Custom MBeans

It is possible to leverage the Catalys management infrastructure to instrument your custom code, such as custom message processors, and expose some attributes in the JMX API. All you need to do is register your MBeans in the platform MBean Server as follows:

```
// Retrieve the platform MBean Server
MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();

// Register an MBean
mbs.registerMBean(new RequiredModelMBean(),
    new ObjectName("MyCompany:name=\"MyCustomMonitor\",type=\"CustomMonitor\"));
```

## F.2. Server Management MBeans

From an operational perspective, the monitoring of the Node focuses on the following information:

- Is your server up and running?
- Should it be up and running?
- Are there any sessions that are down on this server?

Prior to checking the health of the Node, it is useful to check the integrity of the network via this MBean:

```
com.camerontec:name="localhost",type="Host"
```

It has a `ping` operation which can be used to distinguish if the host is alive but the Node is down.

The next step would be to check that the Local Management Agent is running appropriately via this MBean:

```
com.camerontec:Host="<hostName>",type="Host.LMA"
```

Its `live` attribute indicates whether the LMA is currently running.

The information above is normally sufficient to check the integrity of the network link and the presence of an LMA, which in turns allows for monitoring the Node itself.

### F.2.1. Viewing Basic Node Information

The `CatalysServer` MBean provides information on the Node itself, the ability to start and stop the server, and the option to receive notifications of attribute changes. This MBean is found at:

```
com.camerontec:name="<serverName> ",Host="<hostname> ",type="Host.CatalysServer"
```

## Custom Properties

User-defined properties can be used for filtering, which better enables an automated system to determine if the server health is relevant. For instance, defining a custom property for all servers called `criticality` would allow an alert algorithm to distinguish between the failure of a critical server, which may immediately trigger an escalation process, from that of a pre-production server. Custom properties are defined in the server configuration as follows:

```
<Application id="CatalysSell" printMessages="true">
  <Properties id="customProps">
    <Property name="Region" value="Americas"/>
    <Property name="Class" value="Equities"/>
    <Property name="Datacenter" value="Harrison"/>
    <Property name="Escalation" value="(312) 555-5555"/>
  </Properties>
</Application>
```

Other application-level properties will also appear in the MBean, whether defaulted or specifically mentioned in the configuration. For the example above, the `printMessages` attribute will be available.

## Startup Script

The `StartupScript` MBean provides details about the way the Node was started. The MBean can be found at:

```
com.camerontec:CatalysServer="<serverName> ",
Host="<hostname> ",
type="Host.CatalysServer.StartupScript"
```

From this MBean, it's possible to stop the Node process via the `killProcess` operation.

## Managing High Availability Clusters

The `Cluster` and `Cluster.ClusterMember` MBeans provide information HA clusters and their members and provide a way to initiate a failover. The MBeans can be found at:

```
com.camerontec:name="<clusterName> ",type="Cluster"
```

```
com.camerontec:name="<ServerNameInCluster> ",type="Cluster.ClusterMember",Cluster="<clusterName> "
```



## F.3. Session Management MBeans

Session information accounts for some of the most relevant monitoring information provided by the JMX API. From an operational perspective, session monitoring focuses on the following data:

- Connectivity operational data
- FIX protocol operational data
- Session configuration data and properties

Session MBeans are available at:

```
com.camerontec:name="<sessionName>",  
Host="localhost",  
type="Host.CatalysServer.Session",  
CatalysServer="<serverName>"
```

### Connectivity Management

The first aspect of session monitoring relates to the physical and logical connectivity, so as to answer the following questions:

- Is the session connected? (i.e. Transport-level handshake is present)
- Is the session blocked? (i.e. Support has enforced a suspension on this session)
- Is the session in schedule? (i.e. Is it expected to be running at this time?)

The `active`, `status`, `blocked` and `inSchedule` attributes can provide this information.

A number of operations are also exposed, including `reset`, `block` and `unblock`.

### FIX Protocol Management

The `inMsgSeqNum`, `outMsgSeqNum` and `lastResetTime` provide pertinent details about the state of the FIX protocol.

The `reset` and `changeSequenceNumbers` operations can be used to manually adjust the FIX protocol state as necessary.

### Custom Properties

Like the application-level custom properties described earlier, a session can also be configured with such properties. These can be nested as `Properties` under the `Session` element, and are available via the session MBean as the `userDefinedProperties` attribute.

# Appendix G. Third-Party Software

Catalys Node leverages third-party software, both commercial and open source. CameronTec and third-party Java libraries are included in an unbundled format in the /ext and /lib-unbundled directories of the installation. Licenses for third-party software are presented in the following table. Users are invited to review the terms of the licenses.

Library (GroupID:ArtifactID:Version)	License(s)
cglib:cglib-nodep:2.2	<a href="#">The Apache Software License, Version 2.0</a>
ch.qos.logback:logback-classic:1.2.11	<a href="#">GNU Lesser General Public License, version 2.1</a> <a href="#">Eclipse Public License v1.0</a>
ch.qos.logback:logback-core:1.2.11	<a href="#">GNU Lesser General Public License, version 2.1</a> <a href="#">Eclipse Public License v1.0</a>
com.google.code.findbugs:jsr305:3.0.2	<a href="#">The Apache Software License, Version 2.0</a>
com.google.errorprone:error_prone_annotations:-2.18.0	<a href="#">The Apache Software License, Version 2.0</a>
com.google.guava:failureaccess:1.0.1	<a href="#">The Apache Software License, Version 2.0</a>
com.google.guava:guava:32.1.2-jre	<a href="#">The Apache Software License, Version 2.0</a>
com.google.guava:listenablefuture:9999.0-empty-to-avoid-conflict-with-guava	<a href="#">The Apache Software License, Version 2.0</a>
com.google.j2objc:j2objc-annotations:2.8	<a href="#">The Apache Software License, Version 2.0</a>
com.intellij:annotations:12.0	<a href="#">The Apache Software License, Version 2.0</a>
com.lmax:disruptor:3.3.2	<a href="#">The Apache Software License, Version 2.0</a>
com.squareup.javapoet:1.5.1	<a href="#">The Apache Software License, Version 2.0</a>
com.thoughtworks.paranamer:paranamer:2.3	<a href="#">Paranamer License (BSD)</a>
com.thoughtworks.xstream:xstream:1.4.20	<a href="#">XStream License (BSD)</a>
commons-beanutils:commons-beanutils:1.9.4	<a href="#">The Apache Software License, Version 2.0</a>
commons-cli:commons-cli:1.6.0	<a href="#">The Apache Software License, Version 2.0</a>

### Third-Party Software

<b>Library (GroupID:ArtifactID:Version)</b>	<b>License(s)</b>
commons-codec:commons-codec:1.16.0	<a href="#">The Apache Software License, Version 2.0</a>
commons-el:commons-el:1.0	<a href="#">The Apache Software License, Version 2.0</a>
commons-lang:commons-lang:2.6	<a href="#">The Apache Software License, Version 2.0</a>
commons-logging:commons-logging:1.2	<a href="#">The Apache Software License, Version 2.0</a>
cryptix:cryptix-pgp-patched:3.2.0-patch2	<a href="#">Cryptix General License</a>
cryptix:cryptix:3.2.0	<a href="#">Cryptix General License</a>
io.github.lukehutch:fast-classpath-scanner:1.9.17	<a href="#">The MIT License</a>
io.github.x-stream:mxparser:1.2.2	XStream License
javatar:javatar:2.5	<a href="#">Java Tar License (Public Domain)</a>
javax.annotation:javax.annotation-api:1.3.2	<a href="#">Common Development and Distribution License (CDDL) v1.0</a>
javax.management:jmxremote_optional:-1_2_2_03-b34.2	<a href="#">Sun Microsystems, Inc. Binary Code License Agreement (Java(TM) Optional Package)</a>
javax.servlet.jsp:javax.servlet.jsp-api:2.3.1	<a href="#">Common Development and Distribution License (CDDL) v1.1</a>
javax.servlet:javax.servlet-api:3.1.0	<a href="#">Common Development and Distribution License (CDDL) v1.1</a>
javax.servlet:jsp-api:2.0	<a href="#">Common Development and Distribution License (CDDL) v1.1</a>
javax.transaction:javax.transaction-api:1.3	<a href="#">Common Development and Distribution License (CDDL) v1.0</a>
javax.websocket:javax.websocket-client-api:1.0	<a href="#">Common Development and Distribution License (CDDL) v1.1</a>
jaxen:jaxen:1.1.6	<a href="#">Jaxen License</a>
joda-time:joda-time:2.5	<a href="#">The Apache Software License, Version 2.0</a>
net.java.dev.jna:jna-platform:4.1.0	<a href="#">GNU Lesser General Public License, version 2.1</a>

### Third-Party Software

<b>Library (GroupID:ArtifactID:Version)</b>	<b>License(s)</b>
net.java.dev.jna:jna:4.1.0	<a href="#">GNU Lesser General Public License, version 2.1</a>
net.openhft:affinity:2.2	<a href="#">The Apache Software License, Version 2.0</a>
net.openhft:chronicle-algorithms:1.16.0	<a href="#">The Apache Software License, Version 2.0</a>
net.openhft:chronicle-bytes:2.17.4	<a href="#">The Apache Software License, Version 2.0</a>
net.openhft:chronicle-core:2.17.0	<a href="#">The Apache Software License, Version 2.0</a>
net.openhft:chronicle-map:3.17.0	<a href="#">The Apache Software License, Version 2.0</a>
net.openhft:chronicle-threads:2.17.0	<a href="#">The Apache Software License, Version 2.0</a>
net.openhft:chronicle-values:2.16.1	<a href="#">The Apache Software License, Version 2.0</a>
net.openhft:chronicle-wire:2.17.5	<a href="#">The Apache Software License, Version 2.0</a>
net.sf.trove4j:trove4j:3.0.3	<a href="#">GNU Lesser General Public License, version 2.1</a>
org.apache.ant:ant-launcher:1.10.12	<a href="#">The Apache Software License, Version 2.0</a>
org.apache.ant:ant:1.10.12	<a href="#">The Apache Software License, Version 2.0</a>
org.apache.commons:commons-collections4:4.4	<a href="#">The Apache Software License, Version 2.0</a>
org.apache.commons:commons-compress:1.24.0	<a href="#">The Apache Software License, Version 2.0</a>
org.apache.commons:commons-configuration2:-2.8.0	<a href="#">The Apache Software License, Version 2.0</a>
org.apache.commons:commons-lang3:3.12.0	<a href="#">The Apache Software License, Version 2.0</a>
org.apache.commons:commons-math:2.2	<a href="#">The Apache Software License, Version 2.0</a>
org.apache.commons:commons-text:1.9	<a href="#">The Apache Software License, Version 2.0</a>
org.apache.geronimo.specs:geronimo-jms_1.1_spec:1.1.1	<a href="#">The Apache Software License, Version 2.0</a>
org.apache.tomcat:tomcat-api:10.1.14	<a href="#">The Apache Software License, Version 2.0</a>
org.apache.tomcat:tomcat-el-api:10.1.14	<a href="#">The Apache Software License, Version 2.0</a>
org.apache.tomcat:tomcat-jasper-el:10.1.14	<a href="#">The Apache Software License, Version 2.0</a>

### Third-Party Software

Library (GroupID:ArtifactID:Version)	License(s)
org.apache.tomcat:tomcat-jasper:10.1.14	<a href="#">The Apache Software License, Version 2.0</a>
org.apache.tomcat:tomcat-jsp-api:10.1.14	<a href="#">The Apache Software License, Version 2.0</a>
org.apache.tomcat:tomcat-juli:10.1.14	<a href="#">The Apache Software License, Version 2.0</a>
org.apache.tomcat:tomcat-servlet-api:10.1.14	<a href="#">Common Development and Distribution License (CDDL) v1.0</a>
org.apache.tomcat:tomcat-util-scan:10.1.14	<a href="#">The Apache Software License, Version 2.0</a>
org.apache.tomcat:tomcat-util:10.1.14	<a href="#">The Apache Software License, Version 2.0</a>
org.checkerframework:checker-qual:3.33.0	<a href="#">The MIT License</a>
org.codehaus.groovy:groovy-dateutil:3.0.13	<a href="#">The Apache Software License, Version 2.0</a>
org.codehaus.groovy:groovy-jmx:3.0.13	<a href="#">The Apache Software License, Version 2.0</a>
org.codehaus.groovy:groovy:3.0.13	<a href="#">The Apache Software License, Version 2.0</a>
org.codehaus.janino:commons-compiler:3.1.6	<a href="#">New BSD license</a>
org.codehaus.janino:janino:3.1.6	<a href="#">New BSD license</a>
org.codehaus.jettison:jettison:1.5.4	<a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jdt:ecj:3.33.0	<a href="#">Eclipse Public License - v 2.0</a>
org.eclipse.jetty.aggregate:jetty-all:-9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty.http2:http2-client:-9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty.http2:http2-common:-9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty.http2:http2-hpack:-9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty.http2:http2-server:-9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>

### Third-Party Software

Library (GroupID:ArtifactID:Version)	License(s)
org.eclipse.jetty.orbit:javax.activation:- 1.1.0.v201105071233	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty.orbit:javax.mail.glassfish:- 1.4.1.v201005082020	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty.orbit:- javax.security.auth.message:- 1.0.0.v201108011116	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty.orbit:javax.servlet.jsp.jstl:- 1.2.0.v201105211821	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty.orbit:org.eclipse.jdt.core:- 3.8.2.v20130121	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty.toolchain:jetty-schemas:3.1.M0	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty.websocket:javax-websocket-client-impl:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty.websocket:javax-websocket-server-impl:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty.websocket:websocket-api:- 9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty.websocket:websocket-client:- 9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty.websocket:websocket-common:- 9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty.websocket:websocket-server:- 9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty.websocket:websocket-servlet:- 9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-alpn-client:- 9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>

### Third-Party Software

Library (GroupID:ArtifactID:Version)	License(s)
org.eclipse.jetty:jetty-annotations:- 9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-client:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-continuation:- 9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-deploy:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-http:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-io:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-jaspi:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-jmx:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-jndi:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-jsp:9.3.0.M1	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-plus:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-quickstart:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-rewrite:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-security:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-server:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a>

# Third-Party Software

Library (GroupID:ArtifactID:Version)	License(s)
	<a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-servlet:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-servlets:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-util-ajax:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-util:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-webapp:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.eclipse.jetty:jetty-xml:9.4.53.v20231009	<a href="#">Eclipse Public License v1.0</a> <a href="#">The Apache Software License, Version 2.0</a>
org.glassfish.web:javax.servlet.jsp.jstl:1.2.2	<a href="#">Common Development and Distribution License (CDDL) v1.1</a>
org.glassfish.web:javax.servlet.jsp:2.3.2	<a href="#">Common Development and Distribution License (CDDL) v1.1</a>
org.glassfish:javax.el:3.0.0	<a href="#">Common Development and Distribution License (CDDL) v1.1</a>
org.jasypt:jasypt:1.9.3	<a href="#">The Apache Software License, Version 2.0</a>
org.jdom:jdom2:2.0.6.1	<a href="#">JDOM License</a>
org.kohsuke.jetbrains:annotations:9.0	<a href="#">The Apache Software License, Version 2.0</a>
org.ops4j.pax.url:pax-url-aether:2.4.5	<a href="#">The Apache Software License, Version 2.0</a>
org.ow2.asm:asm-commons:9.6	<a href="#">New BSD license</a>
org.ow2.asm:asm-tree:9.6	<a href="#">New BSD license</a>
org.ow2.asm:asm:9.6	<a href="#">New BSD license</a>
org.slf4j:slf4j-api:1.7.25	<a href="#">The MIT License</a>



## Third-Party Software

Library (GroupID:ArtifactID:Version)	License(s)
org.springframework:spring-aop:5.3.27	<a href="#">The Apache Software License, Version 2.0</a>
org.springframework:spring-beans:5.3.27	<a href="#">The Apache Software License, Version 2.0</a>
org.springframework:spring-context:5.3.27	<a href="#">The Apache Software License, Version 2.0</a>
org.springframework:spring-core:5.3.27	<a href="#">The Apache Software License, Version 2.0</a>
org.springframework:spring-expression:5.3.27	<a href="#">The Apache Software License, Version 2.0</a>
org.springframework:spring-jcl:5.3.27	<a href="#">The Apache Software License, Version 2.0</a>
org.xerial.snappy:snappy-java:1.1.10.5	<a href="#">The Apache Software License, Version 2.0</a>
org.yaml:snakeyaml:2.2	<a href="#">The Apache Software License, Version 2.0</a>
stax-utils:stax-utils:20070216	<a href="#">stax-utils License</a>
uk.co.real-logic:sbe:1.0.2-RC2	<a href="#">The Apache Software License, Version 2.0</a>
xerces:xercesImpl:2.12.2	<a href="#">The Apache Software License, Version 2.0</a>
xml-apis:xml-apis:1.4.01	<a href="#">The Apache Software License, Version 2.0</a>
xmlpull:xmlpull:1.1.3.1	Public Domain

# The Apache Software License, Version 2.0

Apache License  
Version 2.0, January 2004  
<http://www.apache.org/licenses/>

## TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

### 1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the

## Third-Party Software

direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable

## Third-Party Software

(except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

- (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
- (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
- (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
- (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of

## Third-Party Software

this License, without any additional terms or conditions.  
Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensors regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensors, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensors provide the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

## Third-Party Software

Licensed under the Apache License, Version 2.0 (the "License");  
you may not use this file except in compliance with the License.  
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software  
distributed under the License is distributed on an "AS IS" BASIS,  
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
See the License for the specific language governing permissions and  
limitations under the License.

## New BSD license

Copyright (c) <YEAR>, <OWNER>  
All rights reserved.

Redistribution and use in source and binary forms, with or without  
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,  
this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright  
notice, this list of conditions and the following disclaimer in the  
documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its  
contributors may be used to endorse or promote products derived from this  
software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE  
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
THE POSSIBILITY OF SUCH DAMAGE.

# Common Development and Distribution License (CDDL) v1.0

COMMON DEVELOPMENT AND DISTRIBUTION LICENSE Version 1.0 (CDDL-1.0)

## 1. Definitions.

1.1. Contributor means each individual or entity that creates or contributes to the creation of Modifications.

1.2. Contributor Version means the combination of the Original Software, prior Modifications used by a Contributor (if any), and the Modifications made by that particular Contributor.

1.3. Covered Software means (a) the Original Software, or (b) Modifications, or (c) the combination of files containing Original Software with files containing Modifications, in each case including portions thereof.

1.4. Executable means the Covered Software in any form other than Source Code.

1.5. Initial Developer means the individual or entity that first makes Original Software available under this License.

1.6. Larger Work means a work which combines Covered Software or portions thereof with code not governed by the terms of this License.

1.7. License means this document.

1.8. Licensable means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. Modifications means the Source Code and Executable form of any of the following:

A. Any file that results from an addition to, deletion from or modification of the contents of a file containing Original Software or previous Modifications;

B. Any new file that contains any part of the Original Software or previous Modification; or

C. Any new file that is contributed or otherwise made available under the terms of this License.

1.10. Original Software means the Source Code and Executable form of computer software code that is originally released under this License.

1.11. Patent Claims means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

## Third-Party Software

1.12. Source Code means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. You (or Your) means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, You includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, control means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

### 2. License Grants.

#### 2.1. The Initial Developer Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof).

(c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software available to a third party under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

#### 2.2. Contributor Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make,

## Third-Party Software

use, sell, offer for sale, have made, and/or otherwise dispose of: (1) Modifications made by that Contributor (or portions thereof); and (2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications available to a third party.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: (1) for any code that Contributor has deleted from the Contributor Version; (2) for infringements caused by: (i) third party modifications of Contributor Version, or (ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or (3) under Patent Claims infringed by Covered Software in the absence of Modifications made by that Contributor.

### 3. Distribution Obligations.

#### 3.1. Availability of Source Code.

Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

#### 3.2. Modifications.

The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

#### 3.3. Required Notices.

You must include a notice in each of Your Modifications that identifies You as the Contributor of the Modification. You may not remove or alter any copyright, patent or trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

#### 3.4. Application of Additional Terms.

You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to



## Third-Party Software

indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

### 3.5. Distribution of Executable Versions.

You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipients rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

### 3.6. Larger Works.

You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

## 4. Versions of the License.

### 4.1. New Versions.

Sun Microsystems, Inc. is the initial license steward and may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

### 4.2. Effect of New Versions.

You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being distributed or otherwise made available under any subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.

### 4.3. Modified Versions.

When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a) rename the license and remove any references to the name of the license steward (except to note that the license differs from this License); and (b) otherwise make it clear that the license contains terms which differ from this License.

## Third-Party Software

### 5. DISCLAIMER OF WARRANTY.

COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN AS IS BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABILITY, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

### 6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as Participant) alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

### 7. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY'S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

## Third-Party Software

### 8. U.S. GOVERNMENT END USERS.

The Covered Software is a commercial item, as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of commercial computer software (as that term is defined at 48 C.F.R. 252.227-7014(a)(1)) and commercial computer software documentation as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or provision that addresses Government rights in computer software under this License.

### 9. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdictions conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use, distribute or otherwise make available any Covered Software.

### 10. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

# Common Development and Distribution License (CDDL) v1.1

COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL - Version 1.1)

## Third-Party Software

### 1. Definitions.

1.1. "Contributor" means each individual or entity that creates or contributes to the creation of Modifications.

1.2. "Contributor Version" means the combination of the Original Software, prior Modifications used by a Contributor (if any), and the Modifications made by that particular Contributor.

1.3. "Covered Software" means (a) the Original Software, or (b) Modifications, or (c) the combination of files containing Original Software with files containing Modifications, in each case including portions thereof.

1.4. "Executable" means the Covered Software in any form other than Source Code.

1.5. "Initial Developer" means the individual or entity that first makes Original Software available under this License.

1.6. "Larger Work" means a work which combines Covered Software or portions thereof with code not governed by the terms of this License.

1.7. "License" means this document.

1.8. "Licensable" means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. "Modifications" means the Source Code and Executable form of any of the following:

A. Any file that results from an addition to, deletion from or modification of the contents of a file containing Original Software or previous Modifications;

B. Any new file that contains any part of the Original Software or previous Modification; or

C. Any new file that is contributed or otherwise made available under the terms of this License.

1.10. "Original Software" means the Source Code and Executable form of computer software code that is originally released under this License.

1.11. "Patent Claims" means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12. "Source Code" means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. "You" (or "Your") means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

## Third-Party Software

### 2. License Grants.

#### 2.1. The Initial Developer Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

- (a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and
- (b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof).
- (c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software available to a third party under the terms of this License.
- (d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

#### 2.2. Contributor Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

- (a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and
- (b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: (1) Modifications made by that Contributor (or portions thereof); and (2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).
- (c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications available to a third party.
- (d) Notwithstanding Section 2.2(b) above, no patent license is granted: (1) for any code that Contributor has deleted from the Contributor Version; (2) for infringements caused by: (i) third party modifications of Contributor Version, or (ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or (3) under Patent Claims infringed by Covered Software in the

## Third-Party Software

absence of Modifications made by that Contributor.

### 3. Distribution Obligations.

#### 3.1. Availability of Source Code.

Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

#### 3.2. Modifications.

The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

#### 3.3. Required Notices.

You must include a notice in each of Your Modifications that identifies You as the Contributor of the Modification. You may not remove or alter any copyright, patent or trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

#### 3.4. Application of Additional Terms.

You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

#### 3.5. Distribution of Executable Versions.

You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipients rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or Contributor. You hereby agree to indemnify the Initial Developer and every Contributor

## Third-Party Software

for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

### 3.6. Larger Works.

You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

## 4. Versions of the License.

### 4.1. New Versions.

Oracle is the initial license steward and may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

### 4.2. Effect of New Versions.

You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being distributed or otherwise made available under any subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.

### 4.3. Modified Versions.

When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a) rename the license and remove any references to the name of the license steward (except to note that the license differs from this License); and (b) otherwise make it clear that the license contains terms which differ from this License.

## 5. DISCLAIMER OF WARRANTY.

COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABILITY, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

## 6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically

## Third-Party Software

if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as "Participant") alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. If You assert a patent infringement claim against Participant alleging that the Participant Software directly or indirectly infringes any patent where such claim is resolved (such as by license or settlement) prior to the initiation of patent infringement litigation, then the reasonable value of the licenses granted by such Participant under Sections 2.1 or 2.2 shall be taken into account in determining the amount or value of any payment or license.

6.4. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

### 7. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY'S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

### 8. U.S. GOVERNMENT END USERS.

The Covered Software is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" (as that term is defined at 48 C.F.R. § 252.227-7014(a)(1)) and "commercial computer software documentation" as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights



## Third-Party Software

clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or provision that addresses Government rights in computer software under this License.

### 9. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdictions conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use, distribute or otherwise make available any Covered Software.

### 10. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

NOTICE PURSUANT TO SECTION 9 OF THE COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL)

The code released under the CDDL shall be governed by the laws of the State of California (excluding conflict-of-law provisions). Any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California and the state courts of the State of California, with venue lying in Santa Clara County, California.

# Cryptix General License

Cryptix General License

Copyright (c) 1995-2005 The Cryptix Foundation Limited.  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are

## Third-Party Software

met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE CRYPTIX FOUNDATION LIMITED AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE CRYPTIX FOUNDATION LIMITED OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# Eclipse Public License v1.0

Eclipse Public License - v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

## 1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

## Third-Party Software

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

### 2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

### 3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

## Third-Party Software

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

### 4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

## Third-Party Software

### 5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

### 6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

### 7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version.

## Third-Party Software

Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

# Eclipse Public License - v 2.0

Eclipse Public License - v 2.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM IS SUBJECT TO THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT.

## 1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial content Distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are Distributed by that particular Contributor.

"Contributor" means any person or entity that Distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of the Program.

"Program" means the Contributions Distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement or any Secondary License (as applicable), including Contributors.

"Derivative Works" shall mean any work, whether in Source Code or other form, that is based on (or derived from) the Program.

"Modified Works" shall mean any work in Source Code or other form that results from an addition to, deletion from, or modification of the Program.

"Distribute" means the acts of a) distributing or b) making available in any manner that enables the transfer of a copy of the Program.

"Source Code" means the form of a Program preferred for making modifications, including but not limited to software source code.

"Secondary License" means either the GNU General Public License, Version 2.0, or any later versions of that license, or any other license that is compatible with the GNU General Public License, Version 2.0.

## 2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free license to:

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free license to:

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no Contributor warrants that the Program is free of infringement.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to make the representations made above.

e) Notwithstanding the terms of any Secondary License, no Contributor makes additional grants to any Recipient (other than those made under this Agreement) by virtue of any act of Distribution of the Program.

## 3. REQUIREMENTS

## Third-Party Software

3.1 If a Contributor Distributes the Program in any form, then:

- a) the Program must also be made available as Source Code, in accordance with section 3.2, and the Contributor must also make available to the Recipient a copy of the Source Code under the same license as the Program;
  - b) the Contributor may Distribute the Program under a license different than this Agreement, provided that such license:
    - i) effectively disclaims on behalf of all other Contributors all warranties and conditions, express and implied, including but not limited to warranties of merchantability and fitness for a particular purpose;
    - ii) effectively excludes on behalf of all other Contributors all liability for damages, including direct, indirect, special, and consequential damages;
    - iii) does not attempt to limit or alter the recipients' rights in the Source Code under section 3.2; and
    - iv) requires any subsequent distribution of the Program by any party to be under a license that satisfies the requirements of this section.
- 3.2 When the Program is Distributed as Source Code:

- a) it must be made available under this Agreement, or if the Program (i) is combined with other material in a separate product, it must be made available under the same license as the Program;
- b) a copy of this Agreement must be included with each copy of the Program.

3.3 Contributors may not remove or alter any copyright, patent, trademark, attribution notices, disclaimers of warranty, or other notices contained in or accompanying the Program.

### 4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and other recipients of the Program.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is responsible for obtaining any applicable licenses for the use of the Program in the product.

### 5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, AND TO THE EXTENT PERMITTED BY APPLICABLE LAW, THE PROGRAM IS PROVIDED "AS IS."

### 6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, AND TO THE EXTENT PERMITTED BY APPLICABLE LAW, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING BUT NOT LIMITED TO PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE PROGRAM, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

### 7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity of the remaining provisions of this Agreement.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program or any portion thereof constitutes an infringement of a patent owned by Recipient, then Recipient's patent litigation shall include this action in order to recover its costs of prosecuting the patent litigation (including any attorney's fees) and any damages it may incur.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement must be distributed on the same terms as this Agreement.

Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, or otherwise.

Exhibit A - Form of Secondary Licenses Notice

"This Source Code may also be made available under the following Secondary Licenses when the conditions for such availability are met:

Simply including a copy of this Agreement, including this Exhibit A is not sufficient to license the Source Code under any of these Secondary Licenses.

If it is not possible or desirable to put the notice in a particular file, then You may include the notice in a location (such as a separate file) where the User can find it.

You may add additional accurate notices of copyright ownership.

## Java Tar License (Public Domain)

## Third-Party Software

---- Public Domain ----

This work was authored by Timothy Gerard Endres, time@gjt.org.

This work has been placed into the public domain.

You are free to use this work in any way you wish.

### DISCLAIMER

THIS SOFTWARE IS PROVIDED AS-IS, WITH ABSOLUTELY NO WARRANTY.  
YOU ASSUME ALL RESPONSIBILITY FOR ANY AND ALL CONSEQUENCES  
THAT MAY RESULT FROM THE USE OF THIS SOFTWARE!

## Jaxen License

```
/*
$Id: LICENSE.txt 1128 2006-02-05 21:49:04Z elharo $

Copyright 2003-2006 The Werken Company. All Rights Reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

* Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions and the following disclaimer in the
  documentation and/or other materials provided with the distribution.

* Neither the name of the Jaxen Project nor the names of its
  contributors may be used to endorse or promote products derived
  from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*/
```



# JDOM License

Copyright (C) 2000-2012 Jason Hunter & Brett McLaughlin.  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution.
3. The name "JDOM" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact <request\_AT\_jdom\_DOT\_org>.
4. Products derived from this software may not be called "JDOM", nor may "JDOM" appear in their name, without prior written permission from the JDOM Project Management <request\_AT\_jdom\_DOT\_org>.

In addition, we request (but do not require) that you include in the end-user documentation provided with the redistribution and/or in the software itself an acknowledgement equivalent to the following:

"This product includes software developed by the  
JDOM Project (<http://www.jdom.org/>)."

Alternatively, the acknowledgment may be graphical using the logos available at <http://www.jdom.org/images/logos>.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE JDOM AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the JDOM Project and was originally created by Jason Hunter <jhunter\_AT\_jdom\_DOT\_org> and Brett McLaughlin <brett\_AT\_jdom\_DOT\_org>. For more information on the JDOM Project, please see <<http://www.jdom.org/>>.

# GNU Lesser General Public License, version 2.1

GNU LESSER GENERAL PUBLIC LICENSE  
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.  
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA  
Everyone is permitted to copy and distribute verbatim copies  
of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts  
as the successor of the GNU Library Public License, version 2, hence  
the version number 2.1.]

## Preamble

The licenses for most software are designed to take away your  
freedom to share and change it. By contrast, the GNU General Public  
Licenses are intended to guarantee your freedom to share and change  
free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some  
specially designated software packages--typically libraries--of the  
Free Software Foundation and other authors who decide to use it. You  
can use it too, but we suggest you first think carefully about whether  
this license or the ordinary General Public License is the better  
strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use,  
not price. Our General Public Licenses are designed to make sure that  
you have the freedom to distribute copies of free software (and charge  
for this service if you wish); that you receive source code or can get  
it if you want it; that you can change the software and use pieces of  
it in new free programs; and that you are informed that you can do  
these things.

To protect your rights, we need to make restrictions that forbid  
distributors to deny you these rights or to ask you to surrender these  
rights. These restrictions translate to certain responsibilities for  
you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis  
or for a fee, you must give the recipients all the rights that we gave  
you. You must make sure that they, too, receive or can get the source  
code. If you link other code with the library, you must provide  
complete object files to the recipients, so that they can relink them  
with the library after making changes to the library and recompiling  
it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the  
library, and (2) we offer you this license, which gives you legal

## Third-Party Software

permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run

## Third-Party Software

that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

### GNU LESSER GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and

## Third-Party Software

distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a) The modified work must itself be a software library.
- b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
- c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
- d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for

## Third-Party Software

that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

## Third-Party Software

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

- a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
- b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.
- c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
- d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
- e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library

## Third-Party Software

facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

- a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
- b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any



## Third-Party Software

patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

### NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY

## Third-Party Software

AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

### END OF TERMS AND CONDITIONS

#### How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>
```

```
This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.
```

```
This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.
```

```
You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the
library `Frob' (a library for tweaking knobs) written by James Random Hacker.
```

```
<signature of Ty Coon>, 1 April 1990
Ty Coon, President of Vice
```

That's all there is to it!

# The MIT License

The MIT License (MIT)

Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

# Paranamer License (BSD)

Copyright (c) 2006 Paul Hammant & ThoughtWorks Inc  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN

## Third-Party Software

CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## stax-utils License

Copyright (c) 2004, Christian Niles, unit12.net  
Copyright (c) 2004, Sun Microsystems, Inc.  
Copyright (c) 2006, John Kristian  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- \* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- \* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- \* Neither the name of the listed copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## Sun Microsystems, Inc. Binary Code License Agreement (Java(TM) Optional Package)

Sun Microsystems, Inc.  
Binary Code License Agreement

## Third-Party Software

READ THE TERMS OF THIS AGREEMENT AND ANY PROVIDED SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT") CAREFULLY BEFORE OPENING THE SOFTWARE MEDIA PACKAGE. BY OPENING THE SOFTWARE MEDIA PACKAGE, YOU AGREE TO THE TERMS OF THIS AGREEMENT. IF YOU ARE ACCESSING THE SOFTWARE ELECTRONICALLY, INDICATE YOUR ACCEPTANCE OF THESE TERMS BY SELECTING THE "ACCEPT" BUTTON AT THE END OF THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL THESE TERMS, PROMPTLY RETURN THE UNUSED SOFTWARE TO YOUR PLACE OF PURCHASE FOR A REFUND OR, IF THE SOFTWARE IS ACCESSED ELECTRONICALLY, SELECT THE "DECLINE" BUTTON AT THE END OF THIS AGREEMENT.

1. LICENSE TO USE. Sun grants you a non-exclusive and non-transferable license for the internal use only of the accompanying software and documentation and any error corrections provided by Sun (collectively "Software"), by the number of users and the class of computer hardware for which the corresponding fee has been paid.

2. RESTRICTIONS. Software is confidential and copyrighted. Title to Software and all associated intellectual property rights is retained by Sun and/or its licensors. Except as specifically authorized in any Supplemental License Terms, you may not make copies of Software, other than a single copy of Software for archival purposes. Unless enforcement is prohibited by applicable law, you may not modify, decompile, or reverse engineer Software. Licensee acknowledges that Licensed Software is not designed or intended for use in the design, construction, operation or maintenance of any nuclear facility. Sun Microsystems, Inc. disclaims any express or implied warranty of fitness for such uses. No right, title or interest in or to any trademark, service mark, logo or trade name of Sun or its licensors is granted under this Agreement.

3. LIMITED WARRANTY. Sun warrants to you that for a period of ninety (90) days from the date of purchase, as evidenced by a copy of the receipt, the media on which Software is furnished (if any) will be free of defects in materials and workmanship under normal use. Except for the foregoing, Software is provided "AS IS". Your exclusive remedy and Sun's entire liability under this limited warranty will be at Sun's option to replace Software media or refund the fee paid for Software.

4. DISCLAIMER OF WARRANTY. UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

5. LIMITATION OF LIABILITY. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. In no event will Sun's liability to you, whether in contract, tort (including negligence), or otherwise, exceed the amount paid by you for Software under this Agreement. The foregoing limitations will apply even if the above stated warranty fails of its essential purpose.

6. Termination. This Agreement is effective until terminated. You may terminate this Agreement at any time by destroying all copies of Software. This Agreement will terminate immediately without notice from Sun if you fail to comply with any provision of this Agreement. Upon Termination, you must destroy all copies of Software.

## Third-Party Software

7. Export Regulations. All Software and technical data delivered under this Agreement are subject to US export control laws and may be subject to export or import regulations in other countries. You agree to comply strictly with all such laws and regulations and acknowledge that you have the responsibility to obtain such licenses to export, re-export, or import as may be required after delivery to you.

8. U.S. Government Restricted Rights. If Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in Software and accompanying documentation will be only as set forth in this Agreement; this is in accordance with 48 CFR 227.7201 through 227.7202-4 (for Department of Defense (DOD) acquisitions) and with 48 CFR 2.101 and 12.212 (for non-DOD acquisitions).

9. Governing Law. Any action related to this Agreement will be governed by California law and controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.

10. Severability. If any provision of this Agreement is held to be unenforceable, this Agreement will remain in effect with the provision omitted, unless omission would frustrate the intent of the parties, in which case this Agreement will immediately terminate.

11. Integration. This Agreement is the entire agreement between you and Sun relating to its subject matter. It supersedes all prior or contemporaneous oral or written communications, proposals, representations and warranties and prevails over any conflicting or additional terms of any quote, order, acknowledgment, or other communication between the parties relating to its subject matter during the term of this Agreement. No modification of this Agreement will be binding, unless in writing and signed by an authorized representative of each party.

JAVA(TM) OPTIONAL PACKAGE

JMX(TM) REMOTE API REFERENCE IMPLEMENTATION, VERSION 1.0.X

SUPPLEMENTAL LICENSE TERMS

These supplemental license terms ("Supplemental Terms") add to or modify the terms of the Binary Code License Agreement (collectively, the "Agreement"). Capitalized terms not defined in these Supplemental Terms shall have the same meanings ascribed to them in the Agreement. These Supplemental Terms shall supersede any inconsistent or conflicting terms in the Agreement, or in any license contained within the Software.

1. Software Internal Use and Development License Grant. Subject to the terms and conditions of this Agreement, including, but not limited to Section 3 (Java Technology Restrictions) of these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license to reproduce internally and use internally the binary form of the Software, complete and unmodified, for the sole purpose of designing, developing and testing your Java applets and applications ("Programs").

2. License to Distribute Software. In addition to the license granted in Section 1 (Software Internal Use and Development License Grant) of these Supplemental Terms, subject to the terms and conditions of this Agreement, including but not limited to, Section 3 (Java Technology Restrictions) of

## Third-Party Software

these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license to reproduce and distribute the Software in binary code form only, provided that you (i) distribute the Software complete and unmodified and only bundled as part of your Programs, (ii) do not distribute additional software intended to replace any component(s) of the Software, (iii) do not remove or alter any proprietary legends or notices contained in the Software, (iv) only distribute the Software subject to a license agreement that protects Sun's interests consistent with the terms contained in this Agreement, and (v) agree to defend and indemnify Sun and its licensors from and against any damages, costs, liabilities, settlement amounts and/or expenses (including attorneys' fees) incurred in connection with any claim, lawsuit or action by any third party that arises or results from the use or distribution of any and all Programs and/or Software.

3. Java Technology Restrictions. You may not modify the Java Platform Interface ("JPI", identified as classes contained within the "java" package or any subpackages of the "java" package), by creating additional classes within the JPI or otherwise causing the addition to or modification of the classes in the JPI. In the event that you create an additional class and associated API(s) which (i) extends the functionality of the Java platform, and (ii) is exposed to third party software developers for the purpose of developing additional software which invokes such additional API, you must promptly publish broadly an accurate specification for such API for free use by all developers. You may not create, or authorize your licensees to create additional classes, interfaces, or subpackages that are in any way identified as "java", "javax", "sun" or similar convention as specified by Sun in any naming convention designation.

4. Trademarks and Logos. You acknowledge and agree as between you and Sun that Sun owns the SUN, SOLARIS, JAVA, JINI, FORTE, and iPLANET trademarks and all SUN, SOLARIS, JAVA, JINI, FORTE, and iPLANET-related trademarks, service marks, logos and other brand designations ("Sun Marks"), and you agree to comply with the Sun Trademark and Logo Usage Requirements currently located at <http://www.sun.com/policies/trademarks>. Any use you make of the Sun Marks inures to Sun's benefit.

5. Source Code. Software may contain source code that is provided solely for reference purposes pursuant to the terms of this Agreement. Source code may not be redistributed unless expressly provided for in this Agreement.

6. Termination for Infringement. Either party may terminate this Agreement immediately should any Software become, or in either party's opinion be likely to become, the subject of a claim of infringement of any intellectual property right.

For inquiries please contact: Sun Microsystems, Inc. 4150 Network Circle,  
Santa Clara, California 95054.  
(LFI#138141/Form ID#011801)

## XStream License (BSD)

## Third-Party Software

Copyright (c) 2003-2006, Joe Walnes  
Copyright (c) 2006-2009, 2011 XStream Committers  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of XStream nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.