Apama FIX Adapter

Innovation Release

Version 10.0

April 2017

APAMA

This document applies to Apama FIX Adapter Version 10.0 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: PAX-FIX-100-20170419

Table of Contents

About this Guide

About this documentation

This guide describes how to configure the Apama FIX Adapter.

Online Information

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at http://documentation.softwareag.com. The site requires Empower credentials. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

You can find product information on the Software AG Empower Product Support website at https://empower.softwareag.com.

To submit feature/enhancement requests, get information about product availability, and download products, go to Products.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the Knowledge Center.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at http://techcommunity.softwareag.com. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

Contacting customer support

You may open Apama Support Incidents online via the eService section of Empower at http://empower.softwareag.com. If you are new to Empower, send an email to `empower@softwareag.com` with your name, company, and company email address to request an account.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Directory at https://empower.softwareag.com/public_directory.asp and give us a call.

1 Introduction to Apama FIX Adapter

This section provides an introduction to the Apama FIX Adapter and its various components.

FIX

FIX (http://www.fixprotocol.org) is the industry standard protocol for the real-time exchange of financial related information and electronic trading. The protocol consists of the following two layers:

- The Session Layer which is concerned with establishing and managing a connection as well as ensuring data integrity, sequencing and addressing
- The Application Layer which is concerned with providing business related services such as market data streaming and order execution

QuickFIX

QuickFIX (http://www.quickfixengine.org) is a fully featured, open source FIX engine. It is currently available for Windows, Linux, Solaris, FreeBSD and Mac OS X.

In essence, QuickFIX takes care of the session layer of FIX and provides a convenient abstraction of the application layer over which host applications provide and use business services.

Apama FIX adapter

The Apama FIX adapter is a set of components based upon the open source QuickFIX library that allows Apama applications to connect to and communicate with FIX compliant systems. The Apama FIX adapter is compatible with the FIX 4.2-4.4, FIX 5.0, FIX 5.0 SP1, FIX 5.0 SP2 specifications.

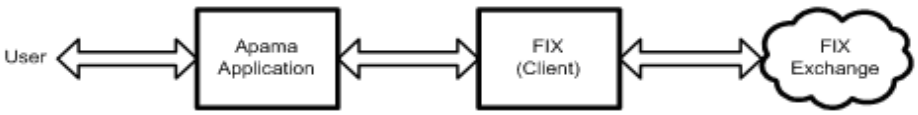
The adapter includes the following components:

- The FIX Transport which links with the QuickFIX library to provide a means of establishing a connections to and exchanging messages with FIX systems
- The Filter Codec which provides the ability to remove fields from FIX messages upon certain conditions
- The FIX Service Monitors which provide a set of business services to Apama applications wishing to use FIX

These components can be composed and configured to act as a FIX client, a FIX order server, or a FIX market data server or a FIX Provider. When used as a FIX client, the FIX adapter provides a means for an Apama application to connect to a FIX compliant exchange and request market data and/or execute trades against its available markets.

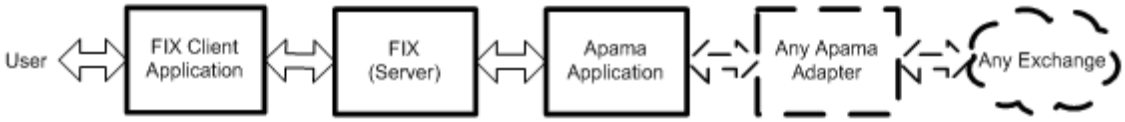
This allows the creation of event based trading applications using the Apama platform. "Figure 1" on page 11 shows a high level diagram of this process:

FIX adapter as a client



When used as a FIX order server, the FIX adapter allows FIX compliant systems to connect to and execute trades against Apama applications. This allows, for example an existing FIX client application to send orders to a scenario which could in turn break those orders up into smaller pieces based on some algorithm and submit them to another completely different exchange. When used as a FIX market data server, the FIX adapter allows FIX compliant systems to connect to and issue requests for market data such as Tick and Depth requests against Apama applications, When used as a FIX provider, the FIX adapter provides a means for a liquidity provider to connect to a FIX compliant exchange.

FIX adapter as a server



2 Release Notes

Release Notes describes the changes introduced with the current Apama FIX Adapter release as well as earlier releases.

What's new in 10.0 release

- Added support for handling Drop Copy sessions.
- Added `SubscriptionManager.RemoveSubscriptionOnReject` to handle MarketDataReject messages that are received when gateway is unavailable (required for TT-FIX).

What's new in 9.12.0 release

- Added support for filtering transaction log messages by FIX message type when "FIXLog.ExcludeMsgTypes" is defined.
- Upgraded QuickFIX library to 1.14.3
- Added message helpers to support an audit logging.
- Added Transport configuration parameter `EnableMessageSendAcknowledgements` to forward the acknowledgements of upstream messages.

What's new in 5.3.1 release

- Added support for Mass Quote stream.

3 Symbol Normalization

The Symbol Normalization feature enables you to use unique symbol (service monitor created symbol) for multi-leg securities. In the absence of this feature, you must give several tag values for each user request (Subscription, NewOrder, AmendOrder, CancelOrder) and must write several "if" conditions to uniquely identify the response event. This new feature reduces the task by providing a `mapped_symbol` so that all the interaction of the application with the service monitor can then be through this `mapped_symbol`. The service monitors will take care of to and fro translation of security definition and mapped symbol.

Session configuration parameters for symbol normalization

Parameter	Description
`DataManager.SymbolFormationTags":"167 207 [146 308 309 310 313]"`	The configuration to create the mapped symbol. This parameter specifies which tag values must be used for constructing the mapped symbol. The mapped symbol will have the tags values in the same sequence as mentioned in the configuration including the position of occurrence of the repeating group.
`DataManager.MappedSecDefListenerKey": "MARKET_DATA"`	This parameters specifies which MarketId/ Transport name to be used for fetching the security information of the mapped symbol. By default each configured session in the service monitors uses its own MarketId/ Connection/TransportName. In dual session MARKET_DATA and TRADE_DATA, MARKET_DATA populates the intermediate cache of the mapped symbol to the security definition and TRADE_DATA session can use cache created by setting the configuration parameter to corresponding MARKET_DATA transport name. In the absence of this parameter setting, user has to send two `InstrumentDataRequest()`, one with MARKET_DATA transport and other with TRADE_DATA transport to populate the `mapped_symbol` cache for each session. **Note:** You must send the `InstrumentDataRequest()` with MARKET_DATA transport/connection to populate the cache of mapped symbol to

Parameter	Description
	security definition before using the cache in other sessions.

Symbol formation

The symbol is formed with the normalized values of tags mentioned in the configuration parameter `DataManager.SymbolFormationTags`. The tags need to be given space delimited. The repeating group tags can be mentioned in "[ ]" square brackets. The first tag in the brackets needs to be the repeating group tag. The tags in the "[ ]" are also space delimited.

From the configuration DataManager.SymbolFormationTags":"167 207 [146 309 310 313]"

The base tags : 167 207

Repeating group tags : 309 310 313

For example:

```
(tag string)
  167:207:309:310:313-309:310:313-309:310:313
```

If you replace the tag numbers with their corresponding values picked from the security definition, that will become the mapped symbol.

```
MLEG:ICE_IPE:217922:FUT:201103-217951:FUT:201108-217923:FUT:201110
 (This security has 3 legs)
```

The mapped symbol contains all the legs of the multi-leg security with configured tag values populated. If you find minimal tags to distinguish each security, then the mapped symbol is recommended.

For example:

```
DataManager.SymbolFormationTags":"207 [146 10456]"
  207:10456-10456
  The mapped symbol will be CME:CLF1-CLG1 (This security has only two legs)
```

Multi-leg symbol formation

All the above examples are the multi-leg scenarios.

Single leg symbol formation

In single leg securities, though the session configuration mentions the repeating group tags, the security definition received from exchange will not have the repeating group. So that part is removed while constructing the mapped symbol.

For example:

```
167:207:309:310:313-309:310:313-309:310:313
```

If you replace the tag numbers with their corresponding values, that will become the mapped symbol.

```
MLEG:ICE_IPE:217922 (repeating group absent in the security defintion
  for single legged security)
```

You must choose minimum base tags such that symbol formed will be unique and simple.

The created mapped symbol will appear in the dictionary field "extraParams" of `InstrumentDataResponse()` as a key value pair.

```
 "MAPPED_SYMBOL" : "MLEG:ICE_IPE:217922"
```

Requesting the security definition for multi-leg securities

Following are the request/response events for getting the security definition:

```
package com.apama.fix
event InstrumentDataRequest {
 string marketId;
 string key;
 string symbol;
 dictionary <string,string> extraParams;
 sequence<string> requiredParams;
}
event InstrumentDataResponse {
 string marketId;
 string key;
 boolean success;
 string errorMessage;
 dictionary <string,string> parameters;
}
```

Depth/Tick Subscription Errors

When the session configuration parameter `SubscriptionManager.ReturnZeroPricesOnError` is set to true, the adapter will send depth/tick events in place of DepthSubscriptionError/TickSubscriptionError. The extraparam of this depth/tick event will have the following values:

- **_ERROR**. The error message (corresponds to DepthSubscriptionError.status)
- **_FAULT**. Flag to indicate whether an unsubscribe is necessary or not (corresponds to DepthSubscriptionError.fault)

Requesting required information

In the extraParams field of the InstrumentDataRequest, you can give the filtering information so that the matching security definitions are received as `InstrumentDataResponse()` for each matched security.

Example: To fetch the security definition for the multi-leg instrument whose leg information is as follows

```
309 : 217922 (tag 309 is UnderlyingSecurityID)
 309 : 217951
com.apama.fix.InstrumentDataRequest
("MARKET_DATA","40","",{"167":"MLEG","207":"ICE_IPE","146":"
[{309:\"217922\"},{309:\"217951\"}]"},["55","48","207"])
```

This request will result in unique/multiple `InstrumentDataResponse()` events. If there are more than one security available with this legs information, then user gets multiple responses. In case of multiple responses the boolean flag "success" of response event is set to false.

Receiving multi-leg information as part of InstrumentDataResponse

The 'parameters' field of the `InstrumentDataResponse()` is populated based on the tag list mentioned in `requiredParams` field of the InstrumentDataRequest. The `requiredParams` sequence field now accepts repeating group as a string.

Example: To receive the repeating field tags 310 311 313, the request has to be

```
com.apama.fix.InstrumentDataRequest
("MARKET_DATA","40","",{"167":"MLEG","207":"ICE_IPE"},
["55","48","207","[146 310 311 313]"])
```

The response would be

```
com.apama.fix.InstrumentDataResponse
   ("MARKET_DATA","40",true,"",{"146":"[{\"310\":\"FUT\",\"311\":
           \"IPE e-Brent\",\"313\":\"201103\"},
   {\"310\":\"FUT\",\"311\":\"IPE e-Brent\",\"313\":\"201108\"}]",
           "207":"ICE_IPE","48":"219924","55":"
   IPE e-Brent","MAPPED_SYMBOL":"MLEG:ICE_IPE:_:ICE_IPE:217922:
           FUT:201103-ICE_IPE:217951:FUT:201108"})
```

Note: Always mention the tags in `requiredParams` field of the `InstrumentDataRequest()`. The tags mentioned in this parameter are the out going tags in each request. The mapped symbol cache is formed against these tags as Mapped symbol.

Using mapped symbol

Subscription management

```
com.apama.marketdata.SubscribeDepth("FIX_SERVICE",
         "MARKET_DATA", "CME:CFL1-CFG1",{})
com.apama.marketdata.UnsubscribeDepth("FIX_SERVICE",
        "MARKET_DATA", "CME:CFL1-CFG1",{})
com.apama.marketdata.Depth("CME:CFL1-CFG1",[],[],[],[],[],{})
 or
com.apama.marketdata.SubscribeTick("FIX_SERVICE", "MARKET_DATA",
        "CME:CFL1-CFG1",{})
com.apama.marketdata.UnsubscribeTick("FIX_SERVICE", "MARKET_DATA",
        "CME:CFL1-CFG1",{})
com.apama.marketdata.Tick("CME:CFL1-CFG1", 123.23, 100000,{})
```

Order management

```
com.apama.oms.NewOrder("19","CME:CFL1-CFG1",35350,"BUY",
       "LIMIT",1,"FIX_SERVICE","","","TT_TRADING","","",{})
 com.apama.oms.AmendOrder("19","FIX_SERVICE","CME:CFL1-CFG1",
       120450,"BUY","LIMIT",2000,{})
 com.apama.oms.CancelOrder("19", "FIX_SERVICE", {})
 com.apama.oms.OrderUpdate("19", "CME:CFL1-CFG1",35350,"BUY",
        "LIMIT",1000,true,true,true,false,false,false,"",1000,
```

```
          1000,1000,35350,35350,"Fill:Partial Fill",{})
```

4 FIX Transport

The FIX IAF transport links with the QuickFIX library to provide connectivity, session management, and event handling to and from a FIX system. It is possible to run several transports within a single IAF process and thereby run several FIX clients and/or servers simultaneously.

Transport properties

Each transport can be configured via a number of properties as outlined below:

QuickFIX properties

The transport has been designed so that it is possible to configure the QuickFIX engine via the IAF configuration file by prepending a transport property with "QuickFIX". For example, the following transport property sets the target host of its embedded QuickFIX engine:

```
<property name="QuickFIX.SocketConnectHost" value="server1.abc.com"/>
```

Any QuickFIX property can be set in this way, a full list of which along with the rules for their use can be found at http://www.quickfixengine.org/quickfix/doc/html/configuration.html.

General transport properties

Static fields

Allows static values to be defined for outgoing (upstream) messages. The definition specifies whether the field is inserted into the header or body, the message type and the tag number. Wildcards ("`*`") can be specified for the message type and tag number.

```
StaticField.[header|body].[msg].[tag]
```

Examples:

```
<property name="StaticField.body.D.1" value="foo"/>
<property name="StaticField.header.*.50" value="bar"/>
```

The first example sets body tag 1 in msg type D (`NewOrderSingle`) to the value `foo` whereas the second example sets header tag 50 in all messages to the value `bar`.

Specific message types have precedence over the wildcard, and existing fields in the message will not be over-ridden. Note that if there are two StaticField entries which update the same tag and message type, the second is ignored.

Header field maps

Maps a header field from an incoming (downstream) FIX message to the payload (`extraParams`) of the event sent to the correlator. The name of the generated payload field will be of the form `Header:<tag>`.

```
MapHeaderField.[msgType].[tag]
```

Examples:

```
<property name="MapHeaderField" value="D.52"/>
<property name="MapHeaderField" value="*.50"/>
```

In upstream messages, you can add this information along with extra parameters, which can be useful when you want to place a tag whose value is dynamic in the header part of the message.

To include a tag in the header part of outgoing (upstream) FIX messages, provide `"Header:<tag>":"<value>"` in the `extraParams` of the event sent to the correlator.

Fatal reject messages

Defines a string that will cause the session to be terminated upon receiving a session level reject containing it.

Example:

```
<property name="FatalRejectMessage" value="Range of messages to
     resend is greater than maximum allowed [2500]."/>
```

In this case, the session will be terminated if the sequence number gap exceeds 2500.

Controlling FIX message replay

The FIX session protocol maintains a unique (within a session) sequence numbering of messages transferred in both directions between the FIX client and server. By default, when a connection is established both the client and server will resend any messages from the same FIX session that have not yet been acknowledged by the other party. This means that all messages sent by either party will be processed even if the session is unexpectedly disconnected (e.g. by a network failure) and later resumed.

However, it is sometimes not desirable for all "old" messages to be processed when a session is resumed. For example, orders submitted immediately before a disconnection but only processed when the session resumes might trade against stale prices. In the event that the FIX server does not automatically reject such stale orders, the FIX transport can be configured not to resend FIX New Order Single messages when the session is established. If the `SendNewOrdersOnInitialResend` transport property is set to true (the default), all unacknowledged messages will potentially be replayed by the transport when the session is established. If this property is set to false, unacknowledged New Order Single messages will not be replayed. All other messages types will still be replayed if requested by the server.

Logging FIX messages

By default, the FIX transport logs all incoming and outgoing FIX messages at DEBUG level. The `LogFixMsgAtInfoLevel` transport property can be used to control this behavior. If the value of this property is false (the default), the transport will log all messages at DEBUG level as usual. If this property is set to true, incoming and outgoing FIX messages will be logged at INFO level instead.

Calculating FIX adapter latency

The FIX transport supports the IAF adapter latency measuring framework introduced in Apama 4.0. Please see the IAF documentation for details of how to configure this feature in general.

If timestamp recording is enabled in the FIX transport, timestamps will be attached to FIX New Order Single, Order Cancel/Replace Request and Execution Report messages, and made available to the adapter service monitors on downstream messages.

If timestamp logging is enabled, the FIX transport will log upstream, downstream and round-trip latency for incoming and outgoing messages. In addition, the service monitors can be configured to add timestamps to outgoing orders and amend/cancel requests, allowing the round-trip latency for an order and subsequent acknowledgement or execution to be calculated. See the details of the OrderManager.LogLatency service monitor parameter for more information on this feature.

Transport properties configuration

Property	Description
QuickFIX.FileStorePath	This parameter controls logging of persisted FIX messages (used for replay and gap recovery). This property must be specified to a valid path. This property accepts relative as well as absolute paths. For example ■ Windows(Absolute):- D:\XYZ\FIX_Log. Directory FIX_Log will be created if it does not exist. ■ Windows(Relative):- Fix_Log. Directory FIX_Log will be created in current directory from where the IAF instance started.
QuickFIX.FileLogPath	This parameter controls logging of all incoming and outgoing FIX messages. If this property is not mentioned, the logging is disabled. This property accepts relative as well as absolute paths.
QuickFIX.SendBufferSize QuickFIX.ReceiveBufferSize	These properties can be used to set the QuickFIX TCP socket send/receive buffer sizes to the number of bytes specified. It defaults to OS specific. For example `<property name="QuickFIX.SendBufferSize" value="1024" />`

Property	Description
	`<property name="QuickFIX.ReceiveBufferSize" value="1024" />`
"ConvertNativeStrings" (boolean, default false)	If true, any STRING-type fields in FIX messages should be converted from the native character encoding UTF-8 for downstream messages and from UTF-8, the native encoding for upstream messages. If false, all strings are assumed to be UTF-8.

"NativeEncoding"
(string, default empty)
	If non-empty, this is the name of the native encoding to use for conversions to and from UTF-8. If not specified, the system encoding should be used - this parameter is to override it per-transport.
	For example
	`<property name="ConvertNativeStrings" value="true" />`
	`<property name="NativeEncoding" value="ISO8859-1" />`

"ExcludeTagsFromMessage" (string, default empty)	This property is used to filter field tags from a given message that needs to be sent to exchange. Field tag be part of header/body/trailer. This property applies to upstream messages. Also field tag cannot be a group tag or tag in group. The format is Message type followed by field tag separated with a space. For example, `BE 43 97 50 553 554`. The above message specifies that from message type 'BE', we need to remove field tags 43, 97, 50, 553, 554 from the message. Each message group should be separated from another message group with a comma D 11 55, BE 43 97 50 553 554 `<property name="ExcludeTagsFromMessage" value="D 11 55, BE 43 97 50 553 554" />`
"MaxSessionLogonAttempts"	Configures the maximum number of session logon attempts. For unlimited attempts, you can configure as -1. Defaults to -1. `<property name="MaxSessionLogonAttempts" value="5" />` Once the session reaches maximum limit, you will be notified using com.apama.fix.NotifyUser event followed by stopping connection. You have to retry by sending com.apama.fix.doLogin event.

Property	Description
ValidateLogonRequests (boolean)	Enable it to authenticate clients in server scenarios. (valid only if QuickFIX.ConnectionType = acceptor). Defaults to false.
LogonValidationTimeout (in milliseconds)	Configure a timeout for client connection validation. Considered only if ValidateLogonRequests is enabled. Defaults to 2000.
StopAcceptorOnWatchdogTimeout	Prevents Acceptor connection from stopping if connection is lost between correlator and IAF. Default is true. For example, `<property name="StopAcceptorOnWatchdogTimeout" value="false" />`
FIXLog.ExcludeMsgTypes	Enable it to exclude writing fix messages to the transaction log on the basis of Message types or message groups. It accepts comma separated values of message types/groups. Supported groups: ■ **ADMIN**. Contains {HeartBeat, TestRequest, ResendReqeust, SequenceReset, Logon, Logout, UserRequest, UserResponse} ■ **MDRESPONSE**. Contains {MarketDataSnapshotFullRefresh, MarketDataIncrementalRefresh, MarketDataRequestReject} ■ **MDREQUEST**. Contains {MarketDataRequest} ■ **QUOTEREQUEST**. Contains {QuoteRequest, RFQRequest} ■ **QUOTERESPONSE**. Contains {Quote, QuoteCancel,QuoteRequestReject, QuoteResponse,MassQuote, MassQuoteAcknowledgement} ■ **ORDERREQUEST**. Contains {NewOrderSingle, OrderCancelReplaceRequest, OrderCancelRequest} ■ **ORDERRESPONSE**. Contains {ExecutionReport, OrderCancelReject} ■ **REJECT**. Contains {Reject, BusinessMessageReject}

Property	Description
	Examples `<property name="FIXLog.ExcludeMsgTypes" value="A,REJECT" /> <property name="FIXLog.ExcludeMsgTypes" value="W,X" /> <property name="FIXLog.ExcludeMsgTypes" value="MDRESPONSE,ORDERREQUEST" />`
ForwardrRefMessageonSessionReject	Enable it to forward the rejected message details on SessionReject. The default value is false.

Client transports

For a transport to act as a FIX client the QuickFIX parameter `ConnectionType` must be set to `initiator`. For example, the following FIX transport is configured to connect to `server1.abc.com` on port 10606 with a `senderCompId` of `CLIENT1`:

```
<transport name="ABC_MARKET_DATA" library="FIXTransport">
  <property name="QuickFIX.ConnectionType" value="initiator" />
  <property name="QuickFIX.SocketConnectHost" value="server1.abc.com"/>
  <property name="QuickFIX.SocketConnectPort" value="10606"/>
  <property name="QuickFIX.SenderCompId" value="CLIENT1" />
  <property name="QuickFIX.TargetCompId" value="SERVER" />
  <property name="QuickFIX.ReconnectInterval" value="60" />
  ...
</transport>
```

When acting as a client, the transport creates and maintains a single FIX session to an external server.

Server transports

For a transport to act as a FIX server the `ConnectionType` must be set to `acceptor`. It is then necessary to create a `TargetCompID` for each client that wishes to connect to the server. For example, the following transport is set up for two clients, `CLIENT1` and `CLIENT2` and is listening for connections on port 10602:

```
<transport name="ABC_MARKET_DATA" library="FIXTransport">
  <property name="QuickFIX.ConnectionType" value="acceptor" />
  <property name="QuickFIX.SocketConnectPort" value="10602"/>
  <property name="QuickFIX.SenderCompId" value="SERVER" />
  ...
  <!--clients -->
  <property name="QuickFIX.TargetCompId" value="CLIENT1" />
  <property name="QuickFIX.TargetCompId" value="CLIENT2" />
</transport>
```

When acting as a server, the transport creates a server capable of maintaining several external client sessions.

Latency measurement

The transport supports the Apama 4.x adapter latency measurement framework. See "The IAF Configuration File" section of the "Developing Adapters" book in the Apama 4.x documentation set for full details on configuring the transport to record high-resolution timestamps on downstream events.

To enable timestamp logging and timestamp propagation to marketdata events, you must pass the SessionConfiguration parameters:

```
"SubscriptionManager.LogLatency":"true"
"OrderManager.LogLatency":"true"
```

Client authentication

When the FIX adapter configured as an order server or market data server, it allows the Apama application to authenticate client sessions. To enable client authentication, set the `ValidateLogonRequests` property in `<transport>` section of the adapter's configuration file to true and create an application to validate client sessions by listening for `com.apama.fix.Logon` request(s) and replying with `com.apama.fix.LogonAuthenticationStatus` events.

When the `ValidateLogonRequests` property is enabled for a server transport, for each logon request from a client it will wait for up to the number of milliseconds specified by the `LogonValidationTimeout` property to receive a `LogonAuthenticationStatus` event from the Apama application. On timeout it rejects logon requests with reason "Logon validation timeout".

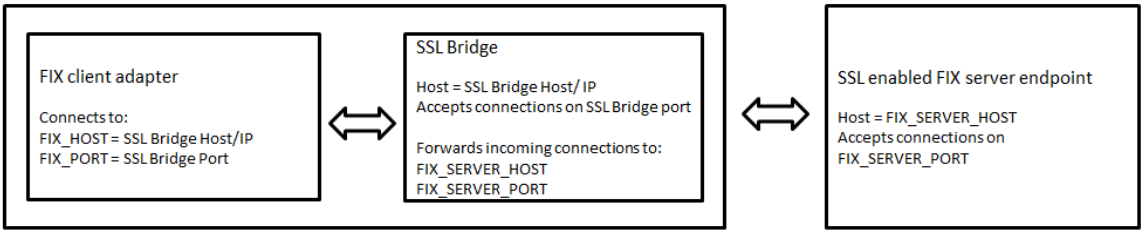
For example, to enable client authentication, the relevant section of the configuration file might look like this:

```
<transport name="ABC_MARKET_DATA" library="FIXTransport">
  <!-- Validate Logon requests, defaults to false -->
  <property name="ValidateLogonRequests" value="true" />
  <!-- Logon validation timeout in milliseconds, defaults to 2000 -->
  <property name="LogonValidationTimeout" value="4000" />
  ...
</transport>
```

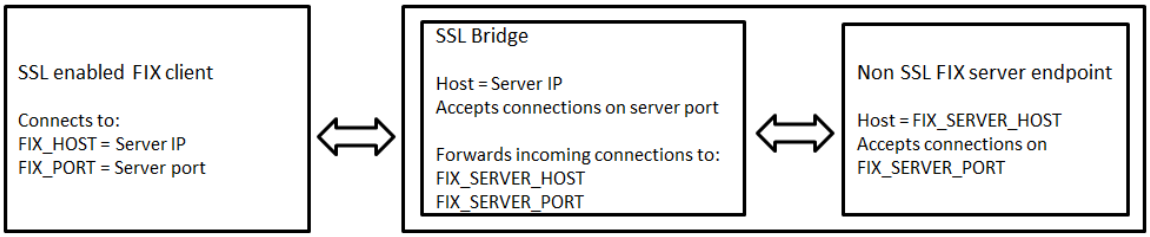
Connecting to SSL enabled FIX servers

The FIX Adapter does not support SSL natively. You can use an SSL wrapper to wrap the FIX server connection inside SSL. For example, Stunnel (www.stunnel.org). The setup will be something like the following:

FIX client connection to SSL enabled FIX adapter



SSL enabled FIX client connecting to FIX server



For example, if the remote server is running on remotehost:443, then in the stunnel configuration file on localhost the following configuration is needed:

```
[fix-adapter]
accept = 12345 (or any unused port number)
connect = remotehost:443
```

Then in the FIX Adapter configuration file, for the transport property `QuickFIX.SocketConnectHost` the value should be "localhost", and for property QuickFIX.SocketConnectPort the value should be "12345".

5 Event Mappings

The IAF configuration files distributed with the FIX adapter come with a standard set of event mappings that include all the messages defined in the FIX 4.2 - 4.4 specifications. Each message maps to a distinct Apama event in the `com.apama.fix` package.

Only fields that are either mandatory in the FIX specification or extremely common are mapped as top level fields. All other fields are mapped into the payload. This also works in reverse in that upstream (outgoing) events can set fields other than those defined as top levels fields by adding them to the payload.

Event routing

In addition to the message fields themselves, each event has two special fields which are used for message routing:

- **transportName** indicates which transport the event has originated from (incoming events) or intended for (outgoing events).
- **SESSION** indicates which session the event has originated from (incoming events) or intended for (outgoing events). For a client transport, this field will always be blank as there is only one session. For a server transport this field indicates the originating or intended client.

Repeating groups

Many FIX messages contain repeating groups of fields which must be mapped to Apama sequences. Each repeating group has a corresponding sub-event defined in the event mapping.

For example, the following event mapping for `MarketDataIncrementalRefresh` defines a repeating group (`NoMDEntries`) which is defined as a sequence of `MarketDataIncrementalRefresh_MDEntry` events:

```
<event name="MarketDataIncrementalRefresh" encoder="FilterCodec"
      direction="both" copyUnmappedToPayload="true" inject="true"
      package="com.apama.fix">
  <id-rules>
    <downstream>
      <id fields="35" test="==" value="X"/>
    </downstream>
    <upstream/>
  </id-rules>
  <mapping-rules>
    <map apama="TRANSPORT" type="string" default="" transport="transportName"/>
    <map apama="SESSION" type="string" default="" transport="SESSION"/>
    <map apama="MDReqID" type="string" default="" transport="262"/>
    <map apama="NoMDEntries" type="reference" referencetype="sequence
         &lt;MarketDataIncrementalRefresh_MDEntry&gt;" default="[]"
           transport="268"/>
    <map type="string" default="X" transport="35"/>
  </mapping-rules>
</event>
<event name="MarketDataIncrementalRefresh_MDEntry" encoder="FilterCodec"
    direction="both" copyUnmappedToPayload="false" inject="true"
```

```
    package="com.apama.fix">
  <id-rules>
    <downstream>
      <id fields="35" test="==" value="_MarketDataIncrementalRefresh_MDEntry"/>
    </downstream>
    <upstream/>
  </id-rules>
  <mapping-rules>
    <map apama="MDUpdateAction" type="string" default="" transport="279"/>
    <map apama="MDEntryType" type="string" default="" transport="269"/>
    <map apama="MDEntryID" type="string" default="" transport="278"/>
    <map apama="Symbol" type="string" default="" transport="55"/>
    <map apama="MDEntryPx" type="float" default="0.0" transport="270"/>
    <map apama="MDEntrySize" type="float" default="0.0" transport="271"/>
    <map apama="OrderID" type="string" default="" transport="37"/>
    <map apama="_extraParams" type="reference" referencetype="dictionary
         &lt;integer, string&gt;" default="{}"/>
    <map type="string" default="_MarketDataIncrementalRefresh_MDEntry"
         transport="35"/>
  </mapping-rules>
</event>
```

Currently, the IAF Normalised Event does not provide a means of representing repeating groups of fields so these sequences must be encoded as strings when passed to or from the transport.

In order for the transport to be able to encode/decode repeating groups it is necessary to specify the structure of each sub-event as a transport property. For example, the following property defines the structure of the `MarketDataIncrementalRefresh_MDEntry` event:

```
<property name="X:268"
          value="com.apama.fix.MarketDataIncrementalRefresh_MDEntry{
    string 279; string 269; string 278; string 55; float 270; float 271;
    string 37; }" />
```

The IAF configuration files distributed with the FIX adapter define properties for all sub-events in the FIX 4.2 specification and a subset of the most useful sub-events in the FIX 4.3 and 4.4 specifications. It is usually not necessary to modify these in any way. However, these properties must be included in any transport that is added. The distributed template configuration files demonstrate the XML "XInclude" syntax used to include the standard repeating group definitions into a FIX adapter configuration file.

6 Filter Codec

The filter codec provides a flexible way of removing fields from events upon certain conditions. This is particularly useful in FIX where they may be certain rules pertaining to event structure in some situations. For example, when sending a market order to a trading system it may be a requirement that no price field is set. With the filter codec is possible to say remove the price field if it is zero.

Each property to the filter codec that begins with `filter.` defines a new filter and takes the form:

```
<property name="filter.<direction>.<field>" value="<value>"/>
```

If no `<field>` is specified then the filter will be applied globally to all fields and likewise for `<direction>`. The `<value>` is what the fields value must be to invoke the filter.

For example, the following configuration removes any field with an empty value in both directions and removes field 44 if its value is "`0`" in the upstream direction.

```
<codec name="FilterCodec" library="FilterCodec">
<property name="removeTransportField" value="false"/>
<property name="TransportFieldName" value="transportName"/>
<property name="filter" value=""/>
  <property name="filter.upstream.44" value="0"/>
</codec>
```

7 FIX Client Monitors

When configured as a client, the FIX adapter enables Apama applications to connect to and interact with external FIX servers. Currently, the FIX adapter provides service monitors that provide support in the areas of market data subscription management, order execution as well as session monitoring and management. For a full list of the areas of FIX that the adapter supports see "Supported FIX Features" on page 89.

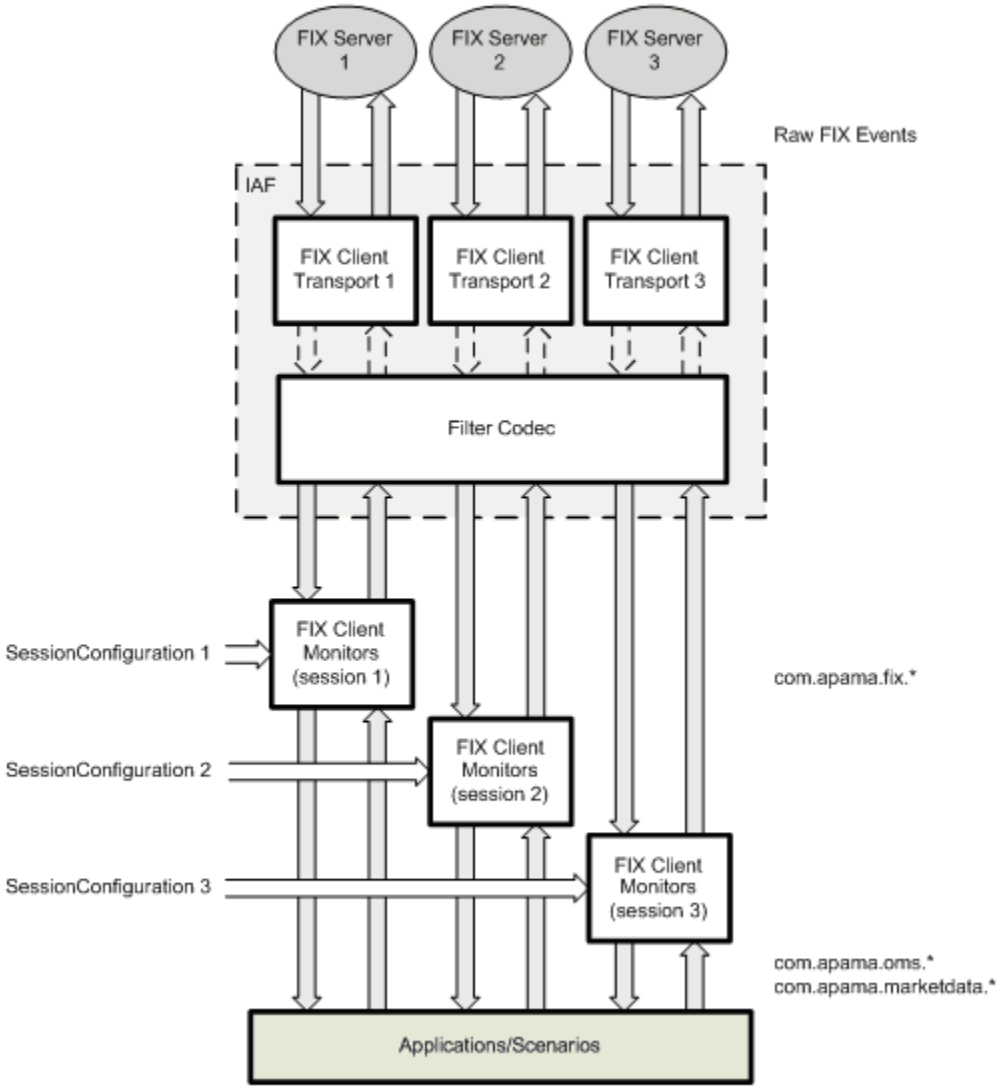
Session configuration

Each service monitor is designed in a way that allows it to be configured differently for each session (i.e. transport) that it is going to interact with. To configure the FIX service monitors for a particular session, the following event must be sent into the correlator:

```
com.apama.fix.SessionConfiguration {
  string connection;                        // FIX transport name
  dictionary<string, string> configuration; //set of configuration parameters
}
```

Each monitor defines it own configuration parameters and sending in this event can be thought of as simultaneously creating a new instance of each service monitor and tailoring its behavior to a particular session. This concept is illustrated below:

FIX adapter as a client



Note, it is not possible to use the FIX service monitors without having first configured them for at least one session.

FIX_SessionManager

The `FIX_SessionManager` provides support to the other monitors by monitoring the state of the FIX sessions which have been configured and sending out notifications when changes occur. The following are monitored:

- Connection Status - The session manager sends a periodic heartbeat request to the transport and expects a reply within 15 seconds. If no such reply is received then all service monitors are notified of a loss of connection for the session in question.
- Session Status - Notifies all service monitors when the transport is logged in or out of the target server.
- Trading Session Status - Some exchanges support the notion of trading session status and report this through the corresponding FIX message. The session manager will notify all service monitors of a change in status.

Configuration parameters

Name	Type	Description
`FixChannel`	`String` Default: "FIX"	The channel to emit upstream events to. See "Session configuration" on page 38.
`SendSessionStatusRequest`	`Boolean` Default: False	Whether to send a FIX `SessionStatusRequest` once logged on.
`NMDA_FIX_SESSION_NAME`	`String`	Provide the SessionName for the NMDA session. This parameter is used to provide the NMDA session name value to the FIX adapter to support session management using the CMF Session Manager library.

FIX_SubscriptionManager

The `FIX_SubscriptionManager` provides market data subscription services via the `com.apama.marketdata` interface (in Software AG Designer, examine the contents of the Finance Bundle for more information.)

The monitor maintains a set of subscriptions, each of which represents the market for a particular instrument or product on an exchange.

Subscribing/unsubscribing

A subscription is created by issuing a `com.apama.marketdata.SubscribeDepth` event specifying the following inputs:

Input	Description
`Symbol`	The instrument (as defined by the exchange)
`ServiceId`	"FIX"
`Market`	The session (i.e. transport) to use
`extraParams`	Any extra parameters to be sent with the request (Note, genuine FIX field names will be automatically translated to their integer tag equivalent)

The subscription manager supports market data through the following FIX mechanisms:

- MarketDataRequest - A `com.apama.marketedata.SubscribeDepth` event will be converted into this type of request by default.
- RequestForQuote - Can be issued instead by setting the extra parameter `Quote=Y`. (Note, only a continuous stream of replacement quotes is supported currently).

The resultant market data updates (or quote stream events) that are generated as a result of the subscription request are converted into `com.apama.marketdata.Depth` and `com.apama.marketdata.Tick` events and routed. Note, each subscription maintains a reference count which is simply incremented for duplicate requests (same symbol + extra parameters) rather than issuing another request to the exchange.

To stop (and completely remove) a subscription, a `com.apama.marketdata.UnsubscribeDepth` event must be routed. This will be translated into the equivalent FIX request and sent to the exchange providing the reference count for the subscription is at zero otherwise the count is decremented.

Extra parameters

As discussed in "Event Mappings" on page 31, any FIX tags which are received from an exchange but are not mapped as top level fields in the event mapping are automatically copied to extra parameters. Since FIX market data messages are composed of a series of repeating entries, the subscription manager associates each parameter with its side and depth level using the following format:

```
(<SIDE><LEVEL>_<NAME>=<VALUE>)
```

For example:

```
"ASK1_Currency=EUR;ASK1_MDEntryOriginator=Bank3;ASK2_Currency=EUR;
ASK2_MDEntryOriginator=Bank1;BID1_Currency=EUR;
BID1_MDEntryOriginator=Bank3"
```

Note, standard FIX tag numbers will be translated to their field names. Where this is not possible (such as for custom tags), the tag will be output as is. For example:

```
"ASK1_9001=12.0;ASK2_9002=foo"
```

Subscription errors

The FIX adapter does not use the `com.apama.marketdata.DepthSubscriptionError` or `com.apama.marketdata.TickSubscriptionError` events to notify clients of a problem with the data stream. Instead, a `com.apama.marketdata.Depth` and/or `com.apama.marketdata.Tick` is issued with `_ERROR` and `_FAULT` set in the extra parameters.

You can use `"_ERROR"` to get the error message (corresponds to `DepthSubscriptionError.status` or `TickSubscriptionError.status`).

`"_FAULT"` indicates the status of the subscription (corresponds to `DepthSubscriptionError.fault` or `TickSubscriptionError.fault`).

Using `Depth` and `Tick` events instead of `DepthSubscriptionError/TickSubscriptionError` allows multiple subscriptions to the same symbol (with differing extra parameters) to be maintained.

Configuration parameters

Name	Type	Description
`FixChannel`	`String` Default: "FIX"	The channel to emit upstream events to. See "Session configuration" on page 38.
`SubscriptionManager.RequireSessionStatusOpen`	`Boolean` Default: False	Whether a FIX session status open message must have been received for the session to be active.
`SubscriptionManager.RequireSessionStatusOpen` `SubscriptionManager.ResubscribeOnConnect`	`Boolean` Default: False	Whether to re-send all subscription requests to the market in the event of a re-connect
`SubscriptionManager.ResubscribeOnLogon`	`Boolean` Default: True	Whether to re-send all subscription requests to the market in the event of a re-logon
`SubscriptionManager.DistinctDepthTickRequest`	`Boolean`	Whether to send separate depth and tick subscription requests

Name	Type	Description
	Default: True	
SubscriptionManager.MarketDataFullRefresh	Boolean Default: False	Whether to request full refreshes (snapshots) rather than snapshot + updates
SubscriptionManager.MarketDataDepthLevel	Integer Default: 0 (Full Book)	Subscription request depth level
SubscriptionManager.FieldOmissions	String Default: ""	Whitespace delimited set of market data entry fields to omit from the output
SubscriptionManager.ForwardMassQuoteRejectReason	boolean Default: False	Enable this to forward the QuoteRejectReason from the MassQuoteAcknowledgement message as the reject reason.
SubscriptionManager.ClearPricesOnSubscribe	Boolean Default: True	Whether to clear all price data prior to issuing a re-subscription.
SubscriptionManager.MaxDepthLevels	Integer Default: 0 (Full Book)	Output depth level
SubscriptionManager.SecDefRequestTags	String Default: ""	Whitespace delimited set of tag numbers to request from the data manager (see "FIX_DataManager" on page 65 for more details)
SubscriptionManager.Aggregate	Boolean Default: False	Whether to aggregate depth entries with the same side and price.
SubscriptionManager.ReuseRequestID	Boolean	Whether to reuse the request id when re-subscribing or generate a new one each time.

Name	Type	Description
	Default: True	
SubscriptionManager. ReturnZeroPricesOnError	Boolean Default: True	Whether to return a Depth event with all prices set to zero instead of a DepthError.Note: The default value of this parameter will change to False in future releases.
SubscriptionManager. IncludeTimeInRequestID	Boolean Default: False	Specifies that the session startup time should be pre-pended to market data request identifiers.
SubscriptionManager. PriceDivisor	Float Default: 1.0	Specifies a value that prices in the depth events send to applications will be divided by.
SubscriptionManager. RecordLatency	Boolean Default: False	Adds support for latency measurement on marketdata messages, stringified timestamp set dictionary is added to com.apama.marketdata events.
SubscriptionManager. LogLatency	Boolean Default: False	Print marketdata latency. Also means that RecordLatency will be enabled.
SubscriptionManager. IgnoreSnapshotOn Unsubscription	Boolean Default: True	Ignores any trade message received after unsubscription.
SubscriptionManager. SupportZeroQuantities	Boolean Default: False	Overwrites old quantity if the new quantity is zero

Name	Type	Description
`SubscriptionManager.MDupdateInPlace`	`Boolean` Default: False	Enables the in-place updating of Market data processing (position based updates are applied in-place).
`SubscriptionManager.SuppressZeroQuantities`	`Boolean` Default: False	Suppress zero quantity entries from `com.apama.marketdata.Depth` event.
`SubscriptionManager.MDUpdateActionOrder`	`Sequence` Default: "1 2 0" (that is, Change, Delete, New)	Sequence of `MDUpdateAction` values separated by spaces. `SubscriptionManager` specified `UpdateAction` order will be used to update marketdata from incremental refresh.
`SubscriptionManager.PublishNonPositivePrices`	`Boolean` Default: False	Publish zero and negative along with positive prices in Depth.
`SubscriptionManager.SubscriptionKeyTags`	`String` Default: ""	Specifies that which parameters or tags should be considered for creating the subscription key for Depth/ Tick subscriptions. The value to be given as string with elements separated by spaces, for example `"22 207"` or `""`. For more about the tags used with this parameter, see "SubscriptionKeyTag note" on page 49.
`SubscriptionManager.UseAltSecurityId`	`Integer` Default: ""	Allows users to put alternative security IDs in the “symbol” field of a market data request and have this mapped to the correct tag on the FIX message (and vice-versa for downstream messages). The value of this should be the number

Name	Type	Description
		of the FIX tag that the alternative security ID will go in. For more information about this parameter, see "UseAltSecurityId note" on page 50.
SubscriptionManager. RepeatingGroupTags	String Default: ""	Repeating groups that are part of outgoing request
SubscriptionManager. UseDefaultTradeEntryTypes	Boolean Default: True	Use the default MDEntry type for Trade Subscription
SubscriptionManager. UseDefaultDepthEntryTypes	Boolean Default: True	Use the default MDEntry type for Depth Subscription
SubscriptionManager. AdditionalMDEntryTypes	String Default: ""	Additional MDEntryTypes to be sent with Subscribe request
SubscriptionManager. RequireSessionStatusOpen	Boolean Default: False	Requires Session State to be open to make subscriptions
SubscriptionManager. SupportZeroQuantities	Boolean Default: False	Overwrites old quantity, if the new quantity is Zero
SubscriptionManager. SupportNonUniqueMDEntryIDs	Boolean Default: False	Required for exchanges which send same MDEntryID for both BID and Offer Sides
SubscriptionManager. QuoteExpireTime	Float Default: 60	Provides the Quote Expiry Time
SubscriptionManager. SubscriptionRequestType	Integer Default: "1"	Provides the Update Mechanism for the subscription (“1” is

Name	Type	Description
		SNAPSHOT_PLUS_ UPDATES)
SubscriptionManager. SendQuoteAck	Boolean Default: False	Send Quote Acknowledgment
SubscriptionManager. delayBeforeResubscription	Float Default: 0	Time to wait before sending a subscription request
SubscriptionManager. UpdateDataWithoutEntryId	Boolean Default: False	Custom handling for updating Order book when EntryID is not present
SubscriptionManager. AcceptNonPositivePrices	Boolean Default: False	Required to support Zero or Negative prices
SubscriptionManager. ForwardMDReqID	Boolean Default: False	Forward MDReqID used in subscription along with Depth updates
SubscriptionManager. UseCurrencyFromSymbol	String Default: “BASE”	Extract the Currency information from the Symbol. Accepted Values are “BASE” and “TERM”. For more information, see "UseCurrencyFromSymbol note" on page 55.
SubscriptionManager. ValidateDepthSubError OnQuoteCancel	Boolean Default: False	SubscriptionManager sends DepthSubscriptionError with 'fault' as "true" in case of QuoteCancel. This parameter is used in QuoteCancel to send DepthSubscription Error.fault as 'false' which indicates that the subscription may still exist and unsubscribe is necessary.

Name	Type	Description
SubscriptionManager. TagsToForwardFromExchange	String Default: ""	List of tags received from exchange to be forward to the user
SubscriptionManager. ReturnZeroPricesOnError	Boolean Default: True	Return empty depth/tick on subscription error
SubscriptionManager. EnableMassQuote Acknowledgement	Boolean Default: False	Enable Mass Quote Acknowledgment
SubscriptionManager. UnsubscribeBeforeSubscribe	Boolean Default: False	Send an unsubscribe request to the exchange before making a subscribe request
SubscriptionManager. GenerateInstrumentID	Boolean Default: False	Lets the adapter generate an instrumentID
SubscriptionManager. IncludeTimeInRequestID	Boolean Default: False	Include Time in Request ID
SubscriptionManager. IgnoreUnexpectedUpdate	Boolean Default: False	Ignore Unexpected Market Data Update
SubscriptionManager. UnsubscribeOnQuoteCancel	Boolean Default: False	Remove subscription when a QuoteCancel is received form exchange
SubscriptionManager. IgnoreQuoteCancel OnUnsubscription	Boolean Default: False	This parameter will handle the QuoteCancel messages received as the response to the Quote Unsubscription.
SubscriptionManager. ClearAttributesOnSnapshot	sequence	This parameter describes what type of prices/quantities

Name	Type	Description
		need to be cleared on receiving a snapshot. It may be either 0 or 1 or 2 . If we assign value * then BID/ OFFER/TRADE/OTHER data is cleared. Defaults to *. If we assign value 0 then BID data is cleared. If we assign value 1 then OFFER data is cleared. If we assign value 2 then TRADE data is cleared. Any value other than */0/1/2 then no data will be cleared

SubscriptionKeyTag note

The tags specified in the `SubscriptionKeyTags` configuration parameter are searched in the same order in the `extraParams` field of `com.apama.marketdata.SubscribeDepth()` and `com.apama.marketdata.SubscribeTick()` to create the subscription key. The reference counting and uniqueness of the subscription is based on presence of these tags and values corresponding to these tags. For example, consider the following:

```
SubscriptionManager.SubscriptionKeyTags = "22 207"
com.apama.marketdata.SubscribeDepth("FIX", "Connection", "SYMBOL",
    {"22":"8", "207":"ABCD", "123":"EFG", "234":"IJK"})
```

The key prepared is:

```
SYMBOL:8:ABCD
```

For the following,

```
com.apama.marketdata.SubscribeDepth("FIX", "Connection", "SYMBOL",
    {"22":"8", "123":"EFG", "234":"IJK"})
```

The key prepared is:

```
SYMBOL:8:
```

The empty value of the configuration parameter indicates that the key is prepared just from the symbol, so the uniqueness is based on just the symbol.

Note: The `SubscriptionManager.SubscriptionKeyTags` configuration parameter is not supported in session reconfiguration, only the initial provided values are used in succeeding re-configurations. Also this parameter has a dependency on the `SubscriptionManager.DistinctDepthTickRequest`

configuration parameter. If both parameters are used in a session configuration event, then for succeeding re-configurations the value of `SubscriptionManager.DistinctDepthTickRequest` should not be altered.

UseAltSecurityId note

You can set the `SubscriptionManager.UseAltSecurityId` and `OrderManager.UseAltSecurityId` configuration parameters to alternate FIX tags; for example:

```
SubscriptionManager.UseAltSecurityId":"10455"
```

In this case a market data request can be sent as:

```
com.apama.marketdata.SubscribeDepth("FIXService", "Transport",
    "AlternateName",{"55":"SYMBOL"})
```

A `NewOrder` can be sent as:

```
com.apama.oms.NewOrder("1040","AlternateName",5.031,"SELL",
    "LIMIT",10,"FIXService","","","Transport","","",{"55":"SYMBOL"})
```

After these are processed by the respective monitors, the `SYMBOL` and `AlternateName` get exchanged and `AlternateName` will be assigned to tag 10455.

FIX_OrderManager

The `FIX_OrderManager` provides order execution services via the `com.apama.oms` interface (in Software AG Designer, examine the contents of the Finance Bundle for more information.)

Placing an order

A new order is placed by issuing a `com.apama.oms.NewOrder` event. The monitor generates a FIX `ClOrdId` and maps this event to a FIX `NewOrderSingle` message and sends it to the specified session. The following inputs are required:

Input	Description
`OrderId`	The block order id seed
`ServiceId`	"FIX"
`Broker`	Unused
`Book`	Unused
`Market`	The session (i.e. transport) to use

Input	Description
`Exchange`	Unused
`Symbol`	The instrument (as defined by the exchange)
`Price`	The price (should be zero for market orders)
`Qty`	The quantity to trade
`Side`	The side 1=BUY, 2=SELL
`Type`	The order type (as defined by the exchange)
`extraParams`	Any extra parameters to be sent with the order (Note, genuine FIX field names will be automatically translated to their integer tag equivalent)

Once in market, the monitor listens for execution reports and translates these to corresponding `com.apama.oms.OrderUpdate` events that describe the current state of the order.

Amending or cancelling an order

To amend or cancel an existing order a `com.apama.oms.AmendOrder` or `com.apama.oms.CancelOrder` event must be routed specifying the order id of the original order. The Order Manager will generate a new FIX client order id and issue a `OrderCancel[Replace]Request` referencing the original client order id.

During a amend or cancel request, the order will be in a non-modifiable state and failures will be notified by setting the `OrderChangeRejected` flag of the `com.apama.oms.OrderUpdate` event.

Trade bust

It is possible in FIX for a previous order execution report to be “undone” by issuing a trade bust. The `FIX_OrderManager` caters for this by providing the `bustTime` configuration parameter which indicates that the execution listeners must be kept “alive” for a certain period of time once the order has been completed.

Configuration parameters

Name	Type	Description
`FixChannel`	`String` Default: "FIX"	The channel to emit upstream events to. See "Session configuration" on page 38.
`OrderManager.RequireSessionStatusOpen`	`Boolean` Default: False	Whether a FIX session status open message must have been received for the session to be active.
`OrderManager.OrderIDServiceName`	`String` Default: "FIX"	The service to request order ids from
`bustTime`	`String` Default: "" (0 days)	Either a decimal number days (relative) or a `cron` format (absolute) time to continue listening for busted trade execution reports after an order has become final.
`OrderManager.SecDefRequestTags`	`String` Default: ""	Whitespace delimited set of tag numbers to request from the data manager (see "FIX_DataManager" on page 65 for more details)
`OrderManager.CopyExecTypeToOrdStatus`	`Boolean` Default: False	Whether to simply assign the value of ExecType to OrdStatus thereby ensuring they are always identical.
`OrderManager.useMillisecond TransactTimes`	`Boolean` Default: False	Whether to use millisecond accuracy on order transact times.
`OrderManager.IgnoreStatusOnOrder CancelReject`	`Boolean` Default: False	Specifies that the status of an order will be unchanged by a `OrderCancelReject` message. The succeeding Amends/Cancels after the rejected message will have their

Name	Type	Description
		`origClorderId` set to clorderId of the last successful order.
`OrderManager.KillOrdersOnSessionDown`	`Boolean` Default: True	Cancels all order listeners when the session goes down.
`OrderManager.RecordLatency`	`Boolean` Default: False	Enables logging of order latency. If this parameter is set to true, the Order Manager will 1) Add a microsecond-accurate timestamp to all outgoing order submissions, amendments and cancellations, and 2) Calculate and log the end-to-end latency of all incoming order executions, assuming that timestamp recording has also been enabled in the transport configuration.
`OrderManager.LogLatency`	`Boolean` Default: False	Print Order management latency, also enables OrderManager.RecordLatency
`OrderManager.CancelRejectUsesOrigId`	`Boolean` Default: False	Handles Order Cancel Reject with swapped `ClOrdID(11)` and `OrigClOrdID(41)` fields.
`OrderManager.amendUsesNonRejectedClOrderID`	`Boolean` Default: False	Specifies that order amend/cancel uses last `ClOrderID` if it is not rejected.
`OrderManager.UseCurrencyFromSymbol`	`String`	Populates the currency (Tag `15`) from supplied symbol. Allowed values are `BASE` and `TERM`. For more information, see "UseCurrencyFromSymbol note" on page 55.
`OrderManager.UseAltSecurityId`	`Integer`	Allows users to put alternative security IDs in the “symbol” field of a market data request and have this mapped to the correct tag on the FIX message (and vice-versa

Name	Type	Description
		for downstream messages). The value of this should be the number of the FIX tag that the alternative security ID will go in. For more information about this parameter, see "UseAltSecurityId note" on page 50.
`FIX.TagsToSupress`	`Sequence` Default: "35 52 34 8"	Provide a list of tags that needs to be removed from the payload of events emitted from the correlator in the upstream direction.
`OrderManager. GenerateNewStyle OrderIds`	`Boolean` Default: False	Generates the id with the complete current (full length) time for uniqueness.
`OrderManager. HandleSuspended ExecType`	`Boolean` Default: False	Required to handle Suspended order ExecType
`OrderManager. CustomOrdTypes`	`String` Default: ""	Supports custom Order Types
`OrderManager. updateKeyParams OnStateChange`	`Boolean` Default: False	Price/Qty fields of order update will update only on an amend order is accepted
`OrderManager. SendExecutionAck`	`Boolean` Default: False	Sends back acknowledgment to Execution Report
`OrderManager. CopyClOrdIDTo OrderUpdate`	`Boolean` Default: False	Forwards ClOrderID sent to the exchange in Order Update
`OrderManager. waitForMarketOrderId`	`boolean`	By enabling this parameter, adapter will not process Cancel/Amend

Name	Type	Description
		order request until NewOrder acknowledged by Exchange.
`OrderManager.SuppressZeroPriceAndQuantity`	`boolean` Default: False	Set this to "true", for exchanges which doesn't send price/quantity for order updates.
`OrderManager.MassCancelUsesMarketId`	`boolean`	Set this to "true", to handle MassOrderCancel based on MarketId.
`OrderManager.SetOrderChangeRejected`	`boolean` Default: False	By enabling this parameter ,the OrderUpdate.orderChangeRejected flag is set to true when order is rejected.
`OrderManager.UseTransactTimeOfOrder`	`boolean`	Use TransactTime forwarded from order.

UseCurrencyFromSymbol note

Use the configuration parameter "OrderManager.UseCurrencyFromSymbol" (For Orders) and “SubscriptionManager.UseCurrencyFromSymbol” (For MarketData Subscriptions) to specify currency pair symbols, for example:

Symbol => USD/GBP or USDGBP (Symbol expected `XXX/YYY` or `XXXYYY`)

BASE currency => USD

TERM currency => GBP

Scenario 1:

In the default behavior (not setting the configuration parameter) you need to give the tag `15` in order requests, without which the order will not be accepted.

Scenario 2:

If you set the configuration parameter and do not specify a value for tag `15` in order requests, the tag will be populated from the symbol as per the configuration.

Scenario 3:

If you set the configuration *and* specify a value for tag `15` in order requests, the supplied tag value will be used.

UseAltSecurityId note

You can set the `SubscriptionManager.UseAltSecurityId` and `OrderManager.UseAltSecurityId` configuration parameters to alternate FIX tags; for example:

```
SubscriptionManager.UseAltSecurityId":"10455"
```

In this case a market data request can be sent as:

```
com.apama.marketdata.SubscribeDepth("FIXService", "Transport",
    "AlternateName",{"55":"SYMBOL"})
```

A `NewOrder` can be sent as:

```
com.apama.oms.NewOrder("1040","AlternateName",5.031,"SELL",
    "LIMIT",10,"FIXService","","","Transport","","",{"55":"SYMBOL"})
```

After these are processed by the respective monitors, the `SYMBOL` and `AlternateName` get exchanged and `AlternateName` will be assigned to tag 10455.

MultiContext in Order Manager

OrderManager now supports MultiContext behaviour. To enable the MultiContext behaviour you must set the `SessionConfig` parameter `OrderManager.MultiContext` to true. You can add/remove contexts dynamically. The user can set up the MultiContext behaviour by either using the MultiContext Helper Utility or it can send the SetupContext request. The creation of the context is the responsibility of the user.

MultiContextAPI

The following events are available:

```
event SetupContext {
    string marketId;
    string requestId;
    context ctx;
   }
```

Used to register/initialize a context. This event should be sent to the Main Context. Parameters are as follows:

- **requestId**. A unique identifier for this request. A `SetupContextResponse` will be received with the same id.
- **ctx**. The context user wants to register.

Once the context is registered, the API will respond with a `SetupContextResponse`.

```
event SetupContextResponse {
    string marketId;
    string requestId;
    context mainCtx;
   }
```

A response event that will be received in the newly created context once the context is successfully registered. The `requestId` will be the same as the one passed in the `SetupContext` event. A reference to the main context is sent in this response.

```
event ClearContext {
    string marketId;
    string requestId;
    context ctx;
   }
```

Used to unregister/uninitialize a context. This event should be sent to the Main Context. The API will respond with a clear context response.

```
event ClearContextResponse {
    string marketId;
    string requestId;
    string success;
   }
```

A response event that will be received in the context that was cleared, once the context has been removed. The parameter success will be true only when there are no pending orders in this context.

Example usage:

```
context mainContext;
  action onload(){
   mainContext := context.current();
   //create the context
   context c := context("dummy", false);
   spawn setup to c;
  }
  action setup(){
   string id := "request1";
   //register this context with the API.
   enqueue com.apama.fix.SetupContext(marketId, id, context.current()) to mainContext;
   on com.apama.fix.SetupContextResponse(marketId=marketId, requestId=id){
    //context registered succesfully
    //send orders
   }
  }
```

MultiContextHelper

A set of actions user that can use to add/remove contexts. The following actions are available:

```
action init(string marketId, context refContext)
```

Used to register a context with the MutliContext API. You must call this action from the newly created context. The parameters are as follows:

- **marketId**. The transport name.
- **refContext**. Should be the Main Context.

> **Note:** `refContext` should be an already existing context, otherwise this context will not be registered. When the context is getting registered, you can send Orders to this context. These orders will be queued and will be sent once the initialization is successful. Upon successful creation of the context, a `SetupContextResponse` event will be sent to the newly created context.

```
action deInit(string marketId, context refContext)
```

Used to unregister a context. Should be called only after `init`. `refContext` should be the Main Context.

Example usage:

```
context mainContext;
  action onload() {
   mainContext := context.current();
   //create a new context
   context c := context("dummy", false);
   spawn setup to c;
  }
  action setup(){
   //register this context
   (new com.apama.fix.MultiContextHelper).init(marketId, mainContext);
   //send orders
  }
```

You can set up the MultiContext behaviour by either sending the `SetupContext` event or by using the helper utility. The helper utility also enables queuing of orders, that is you do not have to wait for a `SetupContextResponse`. It can send orders as soon as it calls the `init`.

As part of the multi-context implementation, the order manager design is moved to use the modular interfaces and callbacks so that the multi-context implementation for the support monitors can be easily extended.

Advantages:

- Easily provide multi-context support for the venue specific monitors.
- Performance improvement by replacing “route event” (if applicable) with callbacks.
- User can provide own implementation by overriding the interface actions.
- Performance improvements due to replacement of nested listener by global listeners.
- Can be easily plugged to other module (multi context, support monitor).

Event Interfaces APIs

Package com.apama.fix

Order Manager(OM) provides the following interfaces and factory interfaces:

OrderManagerInterface

Provides the callback actions to handle the `com.apama.oms` New/Amend/Cancel events and session reconfiguration.

```
event OrderManagerInterface
 {
   //    action call back for handle Neworder
   action<com.apama.oms.NewOrder /*newOrder*/, dictionary<integer, float>
           /*timestamps*/ > onNewOrder;
   //    action call back for handle AmendOrder
   action<com.apama.oms.AmendOrder /*amendOrder*/ , dictionary<integer, float>
            /*timestamps*/ > onAmendOrder;
```

```
   //    action call back for handle CancelOrder
   action<com.apama.oms.CancelOrder /cancelOrder*/ , dictionary<integer, float>
            /*timestamps*/ > onCancelOrder;
   //    action call back for handle Mass CancelOrder
   action<com.apama.oms.CancelOrder /*massCancel*/ > CancelAllOrders;
   //    action call back for Session configuration
   action<SessionConfiguration /*config*/> handleSessionReconfigurations;
  // action to set a callback for order update
     action < action<com.apama.oms.OrderUpdate /*update*/, boolean /*isFinial*/,
         dictionary<integer, float> /*timeStamp*/ > > setOnOrderUpdateCallback;
  //following actions are for internal use only
        action<ContextDictInterface, MultiContextAPI> contextInit;
        action<> reset;
    }
```

Description:

```
   action<com.apama.oms.NewOrder /*order*/, dictionary<integer, float>
     /*timestamps*/ > onNewOrder : This action will be called when Neworder
     event listener is triggered.
   Argument :
      @1 - com.apama.oms.NewOrder event object
      @2 - timestamps dictionary
   action<com.apama.oms.AmendOrder /*amendOrder*/, dictionary<integer, float>
     /*timestamps*/ > onAmendOrder : This action will be called when AmendOrder
     event listener is triggered.
   Argument :
      @1 - com.apama.oms.AmendOrder event object
      @2 - timestamps dictionary
   action<com.apama.oms.CancelOrder /cancelOrder*/, dictionary<integer, float>
      /*timestamps*/ > onCancelOrder : This action will be called when
       CancelOrder event listener is triggered.
   Argument :
      @1 - com.apama.oms.CancelOrder event object
      @2 - timestamps dictionary
   action<com.apama.oms.CancelOrder /*massCancel*/> CancelAllOrders :
    This action will be called when CancelOrder(for MassCancel)
    event listener is triggered.
   Argument :
      @1 - com.apama.oms.CancelOrder event object
   action<SessionConfiguration /*config*/> handleSessionReconfigurations :
    This action is used to handle the session re-configuration.
   Argument :
      @1 - SessionConfiguration event object
```

The response for all placed Orders must be updated by `setOnOrderUpdateCallback` callback.

```
action < action<com.apama.oms.OrderUpdate /*update*/, boolean /*isFinial*/,
 dictionary<integer, float> /*timeStamp*/ > > setOnOrderUpdateCallback :
Argument :
   @1 - An update action that will be called whenever an order update is received.
   This action should have the following parameters -
   @1 - com.apama.oms.OrderUpdate event object
   @2 - the Order is finished or not
   @3 - timestamps dictionary
```

See "BaseFIXOrderManager" module in "FIX_OrderManager.mon" file for the default implementation of above call backs. You can provide custom implementation by overriding these actions.

Messagehandleriface

This interface provides a set of callbacks for handling the FIX based (both downstream and upstream) messages.

```
event Messagehandleriface
 {
  //send NewOrdereSingle message
  action <string /*orderId*/,string /*fixSymbol*/,string /*side*/,
     string /*ordType*/,float /*OrderQty*/,
     dictionary <string, string> /*extraParams*/,
       com.apama.oms.NewOrder/*origOrder*/,
     dictionary<integer,float> /*timeStamp*/> sendNewOrderSingleMessage;
  //send OrderCancelReplaceRequest message
  action <string /*origClOrdId*/,string /*ClOrdId*/,string /*marketOrderId*/ ,
     string /*fixSymbol*/,string /*side*/,string /*ordType*/,float /*OrderQty*/,
     dictionary <string, string> /*extraParams*/,
       com.apama.oms.AmendOrder/*amendOrder*/,
     dictionary<integer,float> /*timeStamp*/> sendOrderCancelReplaceRequest;
  //send OrderCancelReplaceRequest message
  action <string /*origClOrdId*/,string /*clOrdId*/,string /*marketOrderId*/,
             string /*fixSymbol*/, string /*side*/,integer /*quantity*/,
             dictionary <string, string> /*extraParams*/,
      dictionary<integer,float> /*timeStamp*/> sendOrderCancelRequest;
  //send sendQuoteResponse message
  action <string /*orderId*/,string /*fixSymbol*/,string /*side*/,
      string /*ordType*/,float /*OrderQty*/,
      dictionary <string, string> /*extraParams*/,
           com.apama.oms.NewOrder/*origOrder*/,
      dictionary <integer,float> /*timeStamp*/> sendQuoteResponse;
  action <string /*clOrderId*/ >sendOrderMassCancelRequest;
  action <string /*fixId*/,string /*preFixId*/ ,OrderCancelReject
          /*orderCancelReject*/, OrderContainer /*orderContainer*/,
          float /*bustTime*/,dictionary <string, string>
        /*orderRejectedIDs*/ > gotOrderChangeReject;
  action <ExecutionReport /*executionReport*/,OrderContainer
          /*orderContainer*/,float /*bustTime*/,
     dictionary <string, string> /*orderRejectedIDs */ ,
     dictionary<string,OrderContainer>
     /*clOrdIdToOrderContainer*/ > gotExecutionReport;
  action < action<com.apama.oms.OrderUpdate, boolean,
          dictionary<integer, float> >,
   action<string /*fixId*/,OrderContainer /*ordContainer*/>
          /*finishedOrder*/ > setUpdateCallbacks;
  // action call back for Session configuration
  action<SessionConfiguration> handleSessionReconfiguration;
  // generate FIX Order ID
  action <> returns string getFIXOrderID;
  action<> reset;
 }
```

Description:

```
action <string /*orderId*/,string /*fixSymbol*/,string
  /*side*/,string /*ordType*/,float /*OrderQty*/,
  dictionary <string, string> /*extraParams*/,
  com.apama.oms.NewOrder/*origOrder*/,
  dictionary<integer,float> /*timeStamp*/> sendNewOrderSingleMessage;
  It is used to create the NewOrderSingle message based on parameters
  and send it to exchange. When OM receives the NewOrder request,
  it will validate the request then call this interface action for
  order handling.
```

```
Argument :
   @1 - string /*orderId*/  -> clOrdId
   @2 - string /*fixSymbol*/  -> symbol
   @3 - string /*side*/       -> order side
   @4 - string /*ordType*/    -> Order type
   @5 - float /*OrderQty*/    -> Order quantity,
   @6 - dictionary <string, string> /*extraParams*/ -> dictionaries
        of extraParams of NewOrders
   @7 - com.apama.oms.NewOrder/*origOrder*/ -> NewOrder object
   @8 - dictionary<integer,float> /*timeStamp*/ -> timestamps
action <string /*origClOrdId*/,string /*ClOrdId*/,string /*marketOrderId*/ ,
  string /*fixSymbol*/,string /*side*/,string /*ordType*/,float /*OrderQty*/,
  dictionary <string, string> /*extraParams*/,
  com.apama.oms.AmendOrder/*amendOrder*/,
  dictionary<integer,float> /*timeStamp*/> sendOrderCancelReplaceRequest;
  This is used to create the OrderCancelReplaceRequest message based on
  parameters and send it to exchange.
  When OM receives the Amend order request, it will validate the request
  then call this interface action for handle the AmendOrder.
Argument :
   @1 - string /*origClOrdId*/ -> original clOrdId
   @2 - string /*ClOrdId*/  -> clOrdId
   @3 - string /*marketOrderId*/  -> market Order Id
   @4 - string /*fixSymbol*/  -> symbol
   @5 - string /*side*/       -> order side
   @6 - string /*ordType*/    -> Order type
   @7 - float /*OrderQty*/    -> Order quantity,
   @8 - dictionary <string, string> /*extraParams*/ ->
        dictionaries of extraParams of Amend order
   @9 - com.apama.oms.AmendOrder/*amendOrder*/ -> AmendOrder object
   @10 - dictionary<integer,float> /*timeStamp*/ -> timestamps
action <string /*origClOrdId*/,string /*clOrdId*/,string
  /*marketOrderId*/,string /*fixSymbol*/,
  string /*side*/,integer /*quantity*/,dictionary <string, string>
  /*extraParams*/,dictionary<integer,float> /*timeStamp*/>
  sendOrderCancelRequest;
  This is used to create the OrderCancelRequest message based on
  parameters and send it to exchange.
  On receiving the Cancel order request, OM validates the request
  then call this interface action for handling the Cancel order.
Argument :
   @1 - string /*origClOrdId*/ -> original clOrdId
   @2 - string /*ClOrdId*/  -> clOrdId
   @3 - string /*marketOrderId*/  -> market Order Id
   @4 - string /*fixSymbol*/  -> symbol
   @5 - string /*side*/       -> order side
   @6 - float /*OrderQty*/    -> Order quantity,
   @7 - dictionary <string, string> /*extraParams*/ ->
        dictionaries of extraParams of order
   @8 - dictionary<integer,float> /*timeStamp*/ -> timestamps
action <string /*orderId*/,string /*fixSymbol*/,string
  /*side*/,string /*ordType*/,float /*OrderQty*/,
  dictionary <string, string> /*extraParams*/,
  com.apama.oms.NewOrder/*origOrder*/,
  dictionary<integer,float> /*timeStamp*/> sendQuoteResponse;
  This is used to create the QuoteResponse message based on parameters
  and send it to exchange.
Argument :
   @1 - string /*orderId*/ ->  clOrdId
   @2 - string /*fixSymbol*/  -> symbol
   @3 - string /*side*/       -> order side
   @4 - string /*ordType*/    -> Order type
   @5 - float /*OrderQty*/    -> Order quantity,
```

```
   @6 - dictionary <string, string> /*extraParams*/ ->
        dictionaries of extraParams of order
   @7 - com.apama.oms.NewOrder/*origOrder*/ -> NewOrder object,
   @7 - dictionary<integer,float> /*timeStamp*/ -> timestamps.
action <string /*origClOrdId*/,string /*clOrdId*/,string
  /*marketOrderId*/,string /*fixSymbol*/,
  string /*side*/,integer /*quantity*/,dictionary <string, string>
  /*extraParams*/,dictionary<integer,float> /*timeStamp*/>
  sendOrderCancelRequest;
  This is used to create the OrderCancelRequest message based on parameters
  and send it to exchange.
  On received the Cancel order request,OM validate the request then
  call this interface action for handling the Cancel order.
Argument :
   @1 - string /*origClOrdId*/ -> original clOrdId
   @2 - string /*ClOrdId*/  -> clOrdId
   @3 - string /*marketOrderId*/  -> market Order Id
   @4 - string /*fixSymbol*/  -> symbol
   @5 - string /*side*/       -> order side
   @6 - float /*OrderQty*/    -> Order quantity,
   @7 - dictionary <string, string> /*extraParams*/ ->
        dictionaries of extraParams of order
   @8 - dictionary<integer,float> /*timeStamp*/ -> timestamps.
action <string /*clOrderId*/ >sendOrderMassCancelRequest;
  This is used to create the OrderMassCancelRequest message
  based on parameters and send it to exchange.
  On receiving the Mass Cancel request, OM validates the request
  then call this interface action for handling the Mass cancel.
Argument :
   @1 - string /*clOrderId*/ -> clOrdId .
action<SessionConfiguration /*config*/> handleSessionReconfiguration;
  It is used to handled the session re-configuration.
Argument :
  @1 - SessionConfiguration /*config*/ -> SessionConfiguration object
action <> returns string getFIXOrderID;
  It is used to generate the FIX order id. By overriding this call back
  user can provide the custom implementation.
action < action<com.apama.oms.OrderUpdate, boolean,
  dictionary<integer, float> >,
  action<string /*fixId*/,OrderContainer /*ordContainer*/>
  /*finishedOrder*/ > setUpdateCallbacks;
  Using above callbacks , user should update the OrderUpdate and
  finishedOrder details to "BaseFIXOrderManager" module which is
  responsible for memory cleaner and/or update the order response
  to user.
action<com.apama.oms.OrderUpdate, boolean, dictionary<integer,
   float> /*onOrderUpdateCallback*/ - > Order Update call back.
   Argument :
   @1 - com.apama.oms.OrderUpdate /*update*/ -> Order Update object
   @2 - boolean /*isFinial*/ -> is Order finished ?
   @3 - dictionary<integer, float> -> timeStamp.
   action<string /*fixId*/,OrderContainer /*ordContainer*/>
   /*finishedOrder*/ - > finishedOrder
   Argument :
   @1 - string /*fixId*/ -> clOrdId
   @2 - OrderContainer /*container*/ -> Order Container list
```

Downstream Message handle callbacks

```
action <string /*fixId*/,string /*preFixId*/, OrderCancelReject
 /*orderCancelReject*/, OrderContainer /*orderContainer*/,float
 /*bustTime*/,dictionary <string, string> /*orderRejectedIDs*/ >
 gotOrderChangeReject;
```

```
 On receiving the OrderChangeReject request, OM calls this interface action
 to process the Order amend/cancel reject message and update the
 result by calling onOrderUpdateCallback.
Argument :
   @1 - string /*fixId*/ ->  clOrdId
   @2 - string /*preFixId*/  -> original clOrdId
   @3 - OrderCancelReject /*orderCancelReject*/  -> orderCancelReject object
   @4 - OrderContainer /*orderContainer*/ ->
        OrderContainer (it contains corresponding dictionaries)
   @5 - float /*bustTime*/       -> bustTime
   @6 - dictionary <string, string> /*orderRejectedIDs*/ ->
        previous orderRejectedIDs list
action <ExecutionReport /*executionReport*/,OrderContainer
  /*orderContainer*/,float /*bustTime*/,
  dictionary <string, string> /*orderRejectedIDs */ ,
  dictionary<string,OrderContainer> /*clOrdIdToOrderContainer*/ >
  gotExecutionReport;
  On receiving the ExecutionReport request, OM calls this interface action
  to process the execution report message and update the result by calling
  onOrderUpdateCallback.
Argument :
   @1 - ExecutionReport /*executionReport*/ ->  ExecutionReport object
   @2 - OrderContainer /*orderContainer*/  -> OrderContainer
        (it contains corresponding dictionaries)
   @3 - float /*bustTime*/       -> bustTime
   @4 - dictionary <string, string> /*orderRejectedIDs*/ -> previous
        orderRejectedIDs list
   @5 - dictionary<string,OrderContainer> /*clOrdIdToOrderContainer*/ ->
        dictionary for clOrdId to OrderContainer.
```

Factory API's:(package com.apama.fix)

The factory object performs the following steps and returns corresponding interface to the user.

- Create an interface object
- Create the default handler object
- Maps each callbacks to default handler actions.

DefaultMessagehandlerfactory

This factory object takes `Messagehandleriface` interface as input and maps it to the `DefaultMessagehandler`.

It initializes the interface, maps each actions to the default implementation and returns an interface object (`Messagehandleriface`).

```
action create(SessionConfiguration config ,string idNum,
        OrderManagerUtils orderManagerUtils) returns Messagehandleriface.
  Argument:
  @1 - SessionConfiguration /*config*/ -> session config param
  @2 - string /*idNum*/ -> which is used during FIX order id generation.
  @3 - OrderManagerUtils -> OrderManagerUtils object.
```

BaseFIXOrderManagerFactory

This factory object takes `OrderManagerInterface` interface as input and maps it to the `BaseFIXOrderManager`.

It initializes the interface, maps each actions to the default implementation and returns an interface object (`OrderManagerInterface`).

```
action create(string serviceId,string CONNECTION, SessionConfiguration config,
        OrderManagerUtils orderManagerUtils, Messagehandleriface msgIface )
          returns OrderManagerInterface
  Argument:
  @1 - string /*serviceId*/ -> serviceId
  @2 - string /*CONNECTION*/ -> TransportName
  @3 - SessionConfiguration /*config*/ -> session config param
  @4 - OrderManagerUtils -> Order Manager Utils object.
  @5 - Messagehandleriface -> Messagehandleriface interface object.
```

Example(custom implementation for Amend order)

```
monitor OrderManager
 {
  constant string MONITOR_NAME := "FIX Order Manager";
  string SERVICE_NAME := "FIX";
  SessionConfiguration config;
  // the connection to use
  string CONNECTION;
  //the current id
  string idNum := "0";
  OrderManagerUtils orderManagerUtils;
  action onload()
  {
   orderManagerUtils.init();
   on all unmatched SessionConfiguration():config {
    integer val := integer.parse(idNum)+1;
    idNum:=val.toString();
    spawn newSession;
   }
  }
  action newSession {
   CONNECTION:=config.connection;;
   SERVICE_NAME := config.getConfigString(FIXConfigParams.SERVICEID,SERVICE_NAME);
   Messagehandleriface mhiface := (new DefaultMessagehandlerfactory).
               create(config, idNum, orderManagerUtils);
   OrderManagerInterface iface := (new BaseFIXOrderManagerFactory).
              create(SERVICE_NAME, CONNECTION, config, orderManagerUtils, mhiface);
   iface.onAmendOrder := customAmendOrderHandler;
   GenericOrderManager genOMImp := new GenericOrderManager;
   genOMImp.init(SERVICE_NAME,CONNECTION,iface, config) ;
  }
  action customAmendOrderHandler()
  {
  }
 }
```

Floating point quantities

In order to submit a new order or amend an order with floating point quantity values, the following session configuration parameter must be specified:

```
  "AddQuantityToExtraParams":"true"
```

NewOrder and AmendOrder Examples :

```
com.apama.oms.NewOrder("order1", "gbp/usd", 1.9857, "BUY",
   "LIMIT", 6000000, "TRANSPORT", "", "", "Connection", "", "",
```

```
   {"Quantity":"6000000.45"})
com.apama.oms.AmendOrder("order1","TRANSPORT","gbp/usd",1.9857,"BUY",
   "LIMIT",3000000,{"Quantity":"3000000.34"})
```

For trade report service, additional quantity information can also be found in the `com.apama.oms.OrderUpdate` extraParams.

The keys are as follows:

- Quantity
- QuantityExecuted
- QuantityRemaining
- LastQuantityExecuted

FIX_DataManager

The `FIX_DataManager` provides a means to retrieve and store standing data about the products that can be traded on an exchange through the FIX SecurityDefinitionRequest/ Response mechanism.

When used in conjunction with the Subscription Manager and Order Manger, this can be used to automatically retrieve extra information about a particular instrument prior to submitting a request to the market.

For example, some exchanges require a numerical instrument id to be specified when placing an order rather than a descriptive name of the product. In this case, the tag number of the instrument id would be added to the `OrderManager.SecDefRequestTags` configuration parameter which would cause the Order Manager to issue a request for this information to the Data Manager. The Data Manager would then send a security definition request to the exchange containing all the order parameters and providing it is possible to disambiguate the required product, the data manager will return the instrument id back to the Order Manager ready to be inserted into the order itself. Once the data is retrieved, it is cached for future requests.

It will depend on the target exchange whether or not this feature is necessary and possible.

Configuration parameters

Name	Type	Description
`FixChannel`	`String` Default: "FIX"	The channel to emit upstream events to. See "Session configuration" on page 38.

Name	Type	Description
`DataManager.SecurityRequestType`	`String` Default: 3 (Request List Securities)	The security request type to send to the market (see the FIX specification for more details)
`DataManager.SymbolFormationTags`	`String`	Specifies which tags are to be used for symbol normalization. For more about this parameter, see "Symbol Normalization" in the readme file.
`DataManager.MappedSecDef ListenerKey`	`String`	This parameter may be used in conjunction with the `DataManager.SymbolFormationTags` parameter. For example: "" . Empty strings also could be a possible case where in just symbol will be used for creating subscription key. The unique ness is based on just the symbol.

FIX_StatusManager

The `FIX_StatusManager` provides session status notifications to other applications via the `com.apama.statusreport` interface.

For each session the following are monitored:

- Connection Status - Notifies applications of a loss of connection to the IAF as detected by the `FIX_SessionManager`.
- Session Status - Notifies applications when the transport is logged in or out of the target server.
- Trading Session Status - Some exchanges support the notion of trading session status and report this through the corresponding FIX message. The session manager will notify applications of a change in status.
- Market Data - Warns applications that no market data has been received for a specified length of time.

Any application wishing to receive the reports must route a `com.apama.statusreport.SubscribeStatus` event specifying the following parameters (blank strings are taken as wildcards):

Input	Description
`ServiceId`	"FIX"
`Object`	"Adapter"
`Sub_serviceId`	The connection (transport) prefix
`connection`	The connection (transport) name

The sub-service id can be used to subscribe to groups of related connections. For example, if we had the following transports:

```
EXCHANGE1_MARKET_DATA
EXCHANGE1_TRADING
EXCHANGE2_MARKET_DATA
EXCHANGE2_TRADING
```

We could subscribe to all status relating to exchange 1 by routing the following subscribe event:

```
SubscribeStaus("FIX", "Adapter", "EXCHANGE1", "")
```

We could subscribe to status for all the connections by routing the following:

```
SubscribeStaus("FIX", "Adapter", "", "")
```

Note, the sub-service separator token defaults to "_" but can be overridden at session configuration time (see below).

To stop receiving status events a corresponding `com.apama.statusreport.UnsubscribeStatus` event must be routed.

Configuration parameters

Name	Type	Description	Default
`StatusManager.MarketDataTimeout`	`Float`	The period of time in seconds that no market data must have be received before a warning is sent. A value of 0.0 switches the warnings off.	0.0
`SubscriptionManager.RequireSessionStatusOpen`	`Boolean`	Whether the session requires a FIX session status message to	false

Name	Type	Description	Default
		have been received before being available.	
NMDA_FIX_SESSION_NAME	String	Provide the SessionName for the MDA session. This parameter is used to provide the MDA session name value to the FIX adapter to support session management using the CMF Session Manager library.	""

FIX_EventViewer

The FIX_EventViewer provides a means display certain key event types in a readable format in the service monitor log file for debugging purposes. Each field will occupy its own line and where possible enumerated types such as OrdType will be displayed as a string value rather than an integer. Currently, the following events are supported:

- ExecutionReport
- MarketDataSnapshotFullRefresh
- MarketDataIncrementalRefresh
- SecurityStatus
- News

Configuration parameters

Name	Type	Description	Default
EventViewer.Enable	Boolean	Whether to log the events or not.	False

FIX_Events

Contains all event definitions as specified in the standard IAF configuration file (i.e. FIX.xml.dist).

8 Configuring the FIX adapter to use CMF MDA

The FIX adapter supports the Market Data Architecture (MDA) that is available in the Apama Capital Markets Foundation (CMF). While the adapter currently supports only a few connections over CMF's MDA, support for more feeds will be provided in upcoming enhancements.

The FIX adapter can be started in the following configuration modes:

- **MDA Mode** – In this mode, the adapter relies completely on the MDA libraries for connection and MarketData management. Any interaction with the adapter has to be done using only the MDA components. Any interaction related to Order Management requires a separate FIX Connection.
- **Legacy Mode** – The behavior of the adapter in this mode is same as its behavior in versions 4.3 and prior. That is, a session configuration is required to configure the session and users send subscription requests using the `com.apama.marketdata.SubscribeDepth` and `SubscribeTick` interfaces. Both MarketData Management and Order Management can be achieved using this Session, but the MDA framework will not be used in this configuration mode.
- **Mixed Mode** – Use this configuration when you want to use the MDA Session for market data and the Legacy Session for placing orders, with both of them using a single FIX connection.

The following IAF Transport configuration properties have been added in 5.0 to support interaction of the FIX adapter with MDA, as well as to provide various customizations to the FIX adapter:

- IAF_CHANNEL – As the MDA sources are self-registering components, you must provide the IAF channel as a Transport property to make the correlator components aware of the adapter's presence.
- CorrelatorHBTimeout – The timeout interval after which the connection to the correlator can be considered as stale.
- AdapterConfiguration – The adapter configuration details required by the FIX adapter while registering a source with MDA. This is used to provide the list capabilities supported by that particular FIX session, and the details about the module providing that capability. For the supported configuration options, refer to the file `FIXTransport-BaseFIXLibrary.xml` that is shipped with the adapter.

If the AdapterConfiguration transport property is not provided, then the adapter is considered to be in **Legacy Mode**. As such, you have to inject all the FIX adapter monitors and use the com.apama.fix.SessionConfiguration event to configure the adapter session.

You can choose to configure the adapter to be used in **Mixed Mode** (**MDA** for Market Data Management and **Legacy** for placing Orders), as follows:

1. Inject `FIX_NMDA_StatusManager_Bridge.mon` that is shipped with the adapter installation, as well as all the other Legacy FIX service monitors.
2. Confirm that the `AdapterConfiguration` transport property is set, and that the appropriate configuration is provided.

3. Provide the name of the MDA FIX session in the `com.apama.fix.SessionConfiguration` event using `NMDA_FIX_SESSION_NAME` as the key. For example:

```
com.apama.fix.SessionConfiguration("Connection",{"NMDA_FIX_SESSION_NAME":"FIX"})
```

Service monitor injection order

MDA mode

1. ${APAMA_HOME}/monitors/StatusSupport.mon
2. ${APAMA_FOUNDATION_HOME}ASB/projects/general_infrastructure/Foundation Utilities/Foundation Utilities.cdp
3. ${APAMA_FOUNDATION_HOME}/projects/general_infrastructure/Service Framework/Service Framework.cdp
4. ${APAMA_FOUNDATION_HOME}/ASB/projects/general_infrastructure/Data Views/Data View.cdp
5. ${APAMA_HOME}/adapters/monitors/IAFStatusManager.mon
6. ${APAMA_FOUNDATION_HOME}/ASB/projects/general_infrastructure/General Utils/Utils.cdp
7. ${APAMA_FOUNDATION_HOME}/ASB/projects/general_infrastructure/Status Utils/Status Utils_eventdefs-objects.cdp
8. ${APAMA_FOUNDATION_HOME}/ASB/projects/Session Manager/Session Manager_eventdefs-classes.cdp
9. ${APAMA_FOUNDATION_HOME}/ASB/projects/New Market Data API/New Market Data API_events-classes.cdp
10. ${APAMA_FOUNDATION_HOME}/ASB/projects/New Market Data API/New Market Data API_monitors.cdp
11. ${APAMA_FOUNDATION_HOME}/ASB/projects/general_infrastructure/Status Utils/Status Utils_monitors.cdp
12. ${APAMA_FOUNDATION_HOME}/ASB/projects/Session Manager/Session Manager_monitors.cdp

Legacy Mode adapter

1. ${APAMA_HOME}/monitors/ScenarioService.mon
2. ${APAMA_FOUNDATION_HOME}/ASB/projects/Legacy Finance Support/Legacy Finance Support.cdp
3. ${APAMA_HOME}/monitors/StatusSupport.mon
4. ${APAMA_HOME}/adapters/FIX/monitors/FIX_Events.mon

5. ${APAMA_HOME}/adapters/FIX/monitors/FIX_SessionManager.mon
6. ${APAMA_HOME}/adapters/FIX/monitors/FIX_DataManager.mon
7. ${APAMA_HOME}/adapters/FIX/monitors/FIX_SubscriptionManager.mon
8. ${APAMA_HOME}/adapters/FIX/monitors/FIX_OrderManager.mon
9. ${APAMA_HOME}/adapters/FIX/monitors/FIX_EventViewer.mon
10. ${APAMA_HOME}/adapters/FIX/monitors/FIX_StatusManager.mon

Mixed Mode (MDA for Market Data Management and Legacy for placing Orders)

1. ${APAMA_HOME}/monitors/StatusSupport.mon
2. ${APAMA_FOUNDATION_HOME}ASB/projects/general_infrastructure/Foundation Utilities/Foundation Utilities.cdp
3. ${APAMA_FOUNDATION_HOME}/projects/general_infrastructure/Service Framework/Service Framework.cdp
4. ${APAMA_FOUNDATION_HOME}/ASB/projects/general_infrastructure/Data Views/Data View.cdp
5. ${APAMA_HOME}/adapters/monitors/IAFStatusManager.mon
6. ${APAMA_FOUNDATION_HOME}/ASB/projects/general_infrastructure/General Utils/Utils.cdp
7. ${APAMA_FOUNDATION_HOME}/ASB/projects/general_infrastructure/Status Utils/Status Utils_eventdefs-objects.cdp
8. ${APAMA_FOUNDATION_HOME}/ASB/projects/Session Manager/Session Manager_eventdefs-classes.cdp
9. ${APAMA_FOUNDATION_HOME}/ASB/projects/New Market Data API/New Market Data API_events-classes.cdp
10. ${APAMA_FOUNDATION_HOME}/ASB/projects/New Market Data API/New Market Data API_monitors.cdp
11. ${APAMA_FOUNDATION_HOME}/ASB/projects/general_infrastructure/Status Utils/Status Utils_monitors.cdp
12. ${APAMA_FOUNDATION_HOME}/ASB/projects/Session Manager/Session Manager_monitors.cdp
13. ${APAMA_HOME}/monitors/ScenarioService.mon
14. ${APAMA_FOUNDATION_HOME}/ASB/projects/Legacy Finance Support/Legacy Finance Support.cdp
15. ${APAMA_HOME}/adapters/FIX/monitors/FIX_Events.mon
16. ${APAMA_HOME}/adapters/FIX/monitors/FIX_SessionManager.mon
17. ${APAMA_HOME}/adapters/FIX/monitors/FIX_DataManager.mon

18. ${APAMA_HOME}/adapters/FIX/monitors/FIX_SubscriptionManager.mon
19. ${APAMA_HOME}/adapters/FIX/monitors/FIX_OrderManager.mon
20. ${APAMA_HOME}/adapters/FIX/monitors/FIX_EventViewer.mon
21. ${APAMA_HOME}/adapters/FIX/monitors/FIX_StatusManager.mon
22. ${APAMA_HOME}/adapters/FIX/monitors/FIX_NMDA_StatusManager_Bridge.mon

Note: APAMA_HOME is the location of the directory where Apama is installed and APAMA_FOUNDATION_HOME is the location of the directory where CMF is installed.

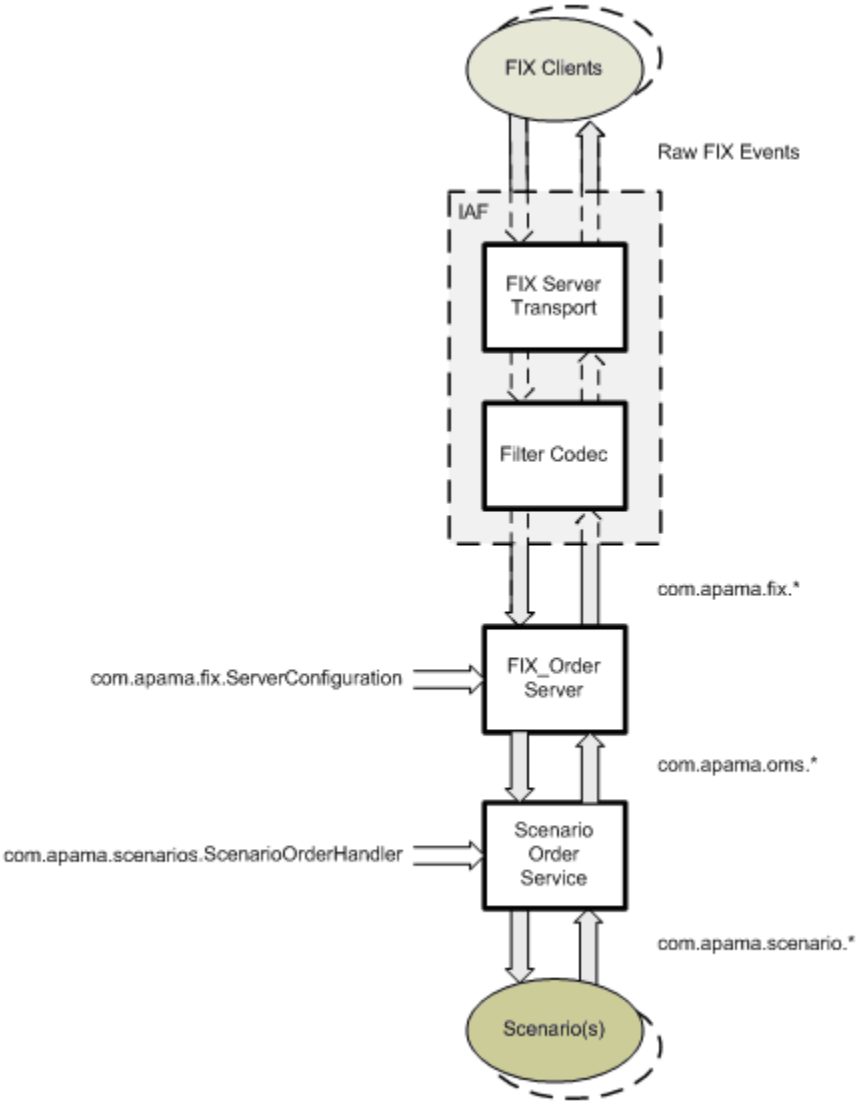
See the README files for specific FIX services for details of any additional service monitors required to use the FIX adapter with those services.

9 FIX Server Monitors

When configured as a server, the FIX adapter enables FIX clients to connect to and interact with Apama applications. When used in conjunction with the scenario order service, It is possible for a FIX client to submit an order to Apama which causes a scenario instance to be created and for that scenario instance to do something with the order and report its status back to the original client via FIX execution reports.

FIX order server

FIX adapter as an order server



In this configuration, the FIX transport is designated as a server (`acceptor`) and `targetCompIds` are set up for each FIX client.

The FIX_OrderServer listens for FIX NewOrderSingle messages and forwards these onto the ScenarioOrderService which instantiates the specified scenario with the values from the order and listens for updates.

Once an order has been created and its scenario has been instantiated, it is possible for the client to modify or delete the order by sending the appropriate FIX messages.

The following sections discuss the FIX_OrderServer and the ScenarioOrderService in detail.

FIX_OrderServer

The FIX_OrderServer translates incoming fix messages (new, amend, cancel) into their apama equivalents and reports on order status using FIX execution reports. An order server is configured and instantiate by sending in the following event:

```
com.apama.fix.ServerConfiguration {
  string connection;
  dictionary<string,string> configuration;
}
```

This event ties the order server to a particular server transport (connection) and sets up its runtime behavior (configuration). It is possible to configure multiple servers if there is more than one server transport configured in the IAF.

On receiving a com.apama.fix.NewOrderSingle event from its transport, the server creates a com.apama.fix.ExecutionReport with a OrdStatus of PENDING_NEW. It then generates an order id and creates a corresponding com.apama.oms.NewOrder event before sending it to the target service. The target service defaults to the scenario order service but can be configured to be something else (for example, a market simulator).

If the original FIX order specifies a target strategy (default FIX tag 847), its identity will be copied into the broker field of the resultant Apama order. This is commonly used to specify a Scenario Order Handler (see "ScenarioOrderService" on page 80). Strategy parameters (default FIX tag 848) will be merged into the extra parameters of the order.

For example, NewOrderSingle should contain the details of the TargetStrategy (Tag 847) and TargetStrategyParameters (Tag 848). TargetStrategyParameters can be provided using the following convention "848="*Param1*:*Value1*;*Param2*:*Value2*; …. ;*ParamN*:*ValueN*", as shown in the following event sample:

```
com.apama.fix.NewOrderSingle("Server","","1:0:1e+09","","1","ORCL","1",
   "20010909-02:46:40",10000,"1",0,[],{},
   {"109":"Client_1_ID","847":"TestBrokerID","848":"strgy1:100;strgy2:200;strgy3:300"})
```

The above NewOrderSingle will be translated to the following:

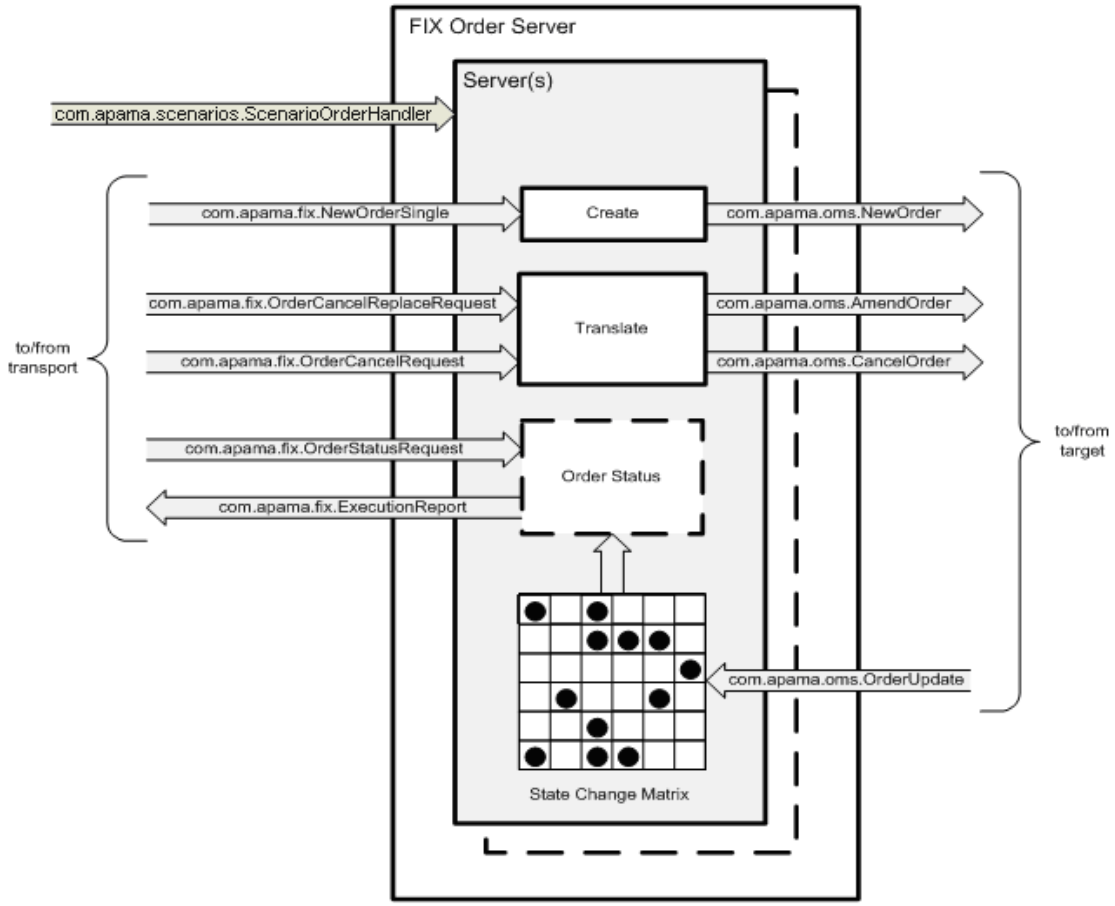
```
com.apama.oms.NewOrder("Server:1e+09:1","ORCL",0,"BUY","MARKET",10000,
   "MarketSimulator","TestBrokerID","","Server","","Client_1_ID",
   {"109":"Client_1_ID","Account":"",
   "__timestamps":"{}","strgy1":"100","strgy2":"200","strgy3":"300"})
```

For each com.apama.oms.OrderUpdate event that is received with generated order id, the order state is determined using an implementation of the state change matrix as defined by the FIX specification (see http://www.fixprotocol.org/specfications).

Any change in state is reported back to the client as a `com.apama.fix.ExecutionReport`.

Order amendments and cancels are possible and are translated into their Apama equivalents (that is, `com.apama.oms.AmendOrder` or `com.apama.oms.CancelOrder`) and it is possible to request order status by sending in a `com.apama.fix.OrderStatusRequest` message for the order in question.

FIX Order Server



Configuration properties

Name	Type	Description
`FixChannel`	`String` Default: FIX	The channel to emit upstream events to. See "Session configuration" on page 38.
`TargetService`	`String` Default: MarketSimulator	The ID of the target service
`ScenarioParameter`	`string` Default: TargetStrategy (847)	The FIX field to extract the scenario ID from
`OrderServer. UseFirewallServiceID`	`boolean`	When set to true, this property adds Firewall service Id in the extraparams while routing NewOrder, AmendOrder and CancelOrder.
`OrderServer. ForwardRequestID`	`boolean`	Enable this to forward a unique requestID in all orders generated by the fix order server. A uniqueID with key as "FIX_SERVER_REQUEST_ID" will be forwarded in oms NewOrder, AmendOrder and CancelOrder. The fix order server expects the tag to be returned in the OrderUpdate with key as "FIX_SERVER_REQUEST_ID_ACK". This parameter is used by the fix order server to identify if the update was generated for the original order or an amend/cancel request.
`OrderServer. RecordLatency`	`boolean`	Adds support for latency measurement on OrderServer.
`OrderServer.`	`boolean`	Print OrderServer latency, also means that

Name	Type	Description
`LogLatency`		OrderServer.RecordLatency will be enabled.
`OrderServer.ValidateSessionParams`	`boolean` Default: False	Set this to "true" to validate incoming OrderUpdates based on SERVICE_NAME and Market.

ScenarioOrderService

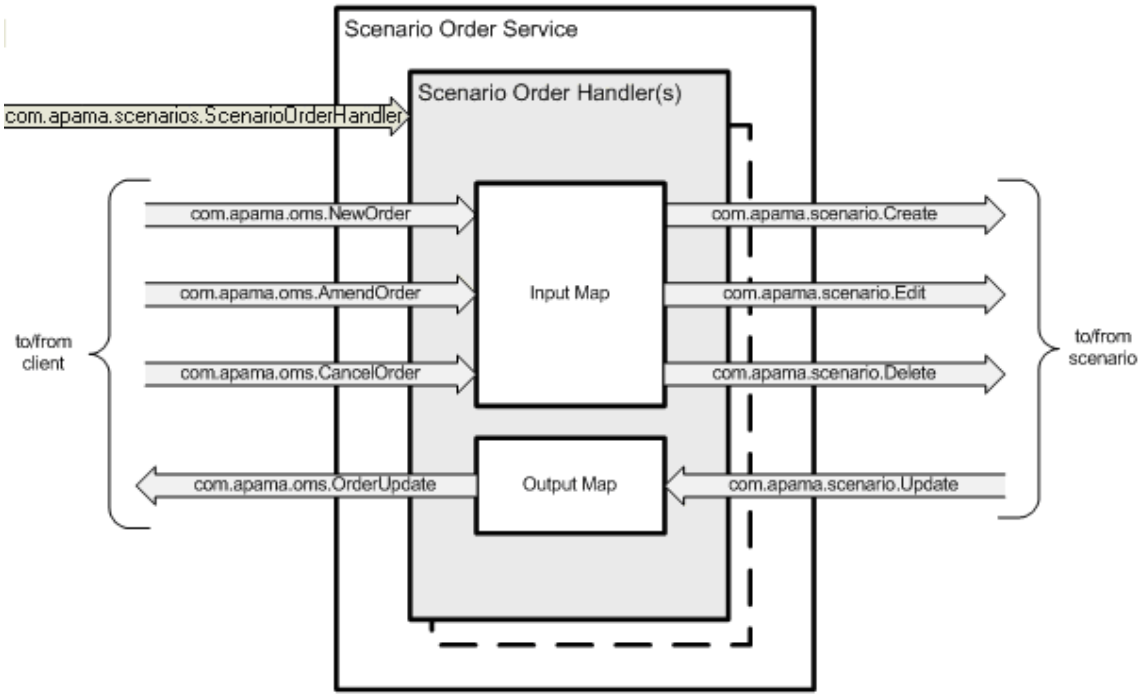
The `ScenarioOrderService` is designed around the notion of a scenario order handler which is an object that translates order events for a particular scenario. The scenario order service can contain many handlers and therefore can manage many scenarios simultaneously. A handler is configured by sending in the following event:

```
com.apama.scenarios.ScenarioOrderHandler {
  string id;
  string scenarioId;
  dictionary<string,string> inputMap;
  dictionary<string,string> outputMap;
  dictionary<string,string> configuration;
}
```

The id allows the handler to be referenced, the `scenarioId` references the scenario to be used, the input and output maps define the mapping between orders and scenario instances (see "Figure 8" on page 81) and the configuration defines the runtime behavior. Note, the scenario referenced by the scenario id must be loaded at the time this event is received as the input/output maps will be validated against the scenario definition.

When a new order event is received the handler id must be specified in the `brokerId` field. If the broker is not specified or the handler cannot be found the order is rejected.

FIX Order Service



Input and output mappings

The input and output mappings determine how the handler translates `com.apama.oms.*` events to and from `com.apama.scenario.*` events. Each mapping is configured as a set of `<target>:<source>` pairs where `<target>` is the field or extra parameter that will be set in the target event and `<source>` is the source field/parameter or an literal value if preceded with a "`#`" character.

For example, the following input map sets the scenario input `orderPrice` to be the order field `price`, the scenario input `orderQuantity` to be the order field `qty` and the scenario input `VWAPWindow` to be the literal value "100" upon receiving a new order:

```
{"orderPrice":"price", "orderQuantity":"qty", "VWAPWindow":"#100"}
```

When the scenario issues a `com.apama.scenario.Update` event its outputs are mapped to a `com.apama.oms.OrderUpdate` event based on the output mapping. For example the following output map sets the order update field `qtyExecuted` to be the scenario output `orderQtyExecuted` and sets the order update field `9101` to be the literal value "101":

```
{"qtyExexcuted":"orderQtyExecuted", "9101":"#101"}
```

The scenario order handler does not distinguish between "real" top level fields and extra parameters. If the named field exists in the target then it will be set, otherwise an extra

parameter of that name will be set. The same works in reverse except that if a source parameter or field doesn't exist, it will be set as blank in the target.

In the case of handling `OrderCancel` events, there are two options. The default option is to simply delete the scenario instance. However this may not be useful in many cases as it does not allow the scenario to do anything before exiting or even reject the cancel request. The second option is to nominate a scenario input (using the configuration parameter `CancelFlag`) to be edited to true thereby allowing the scenario to take appropriate action before exiting.

Configuration properties

Name	Type	Description	Default
`CancelFlag`	`String`	The scenario input to set to true upon receiving a CancelOrder message (must be a boolean input)	`""`

FIX market data server

The FIX_MarketDataServer translates incoming fix market data request messages into their Apama equivalents like `com.apama.mds.SubscribeDepth` and `com.apama.mds.SubscribeTick` events.

A FIX MarketData Server can be configured and instantiated by sending the following event:

```
com.apama.fix.MDServerConfiguration {
    string connection;
    dictionary<string,string> configuration;
}
```

Configuration properties

The following session configuration parameters can be used to configure the FIX market data server.

Name	Type	Description
`RequestKeyParams`	`sequence` Default: * (include all `extraParams`	Specifies the parameters or tags that should be considered for creating the subscription key for Depth/Tick subscriptions. The value is specified as a string with elements separated by spaces.

Name	Type	Description
		■ An asterisk (*) includes all extraParams of MarketDataRequest. ■ An empty string ("") includes none of the extraParams of MarketDataRequest. ■ "336 625" includes tags 336, 625 and their values only. ■ "* 336 625" includes all except tags 336 and 625.
FixChannel	String Default: FIX	The channel to emit upstream events to. See "Session configuration" on page 38.
TargetService	String Default: MarketSimulator	The ID of the target service.
MDS.RecordLatency	boolean	Adds support for latency measurement on MarketDataServer.
MDS.LogLatency	boolean	Print MarketDataServer latency, also means that MDS.RecordLatency will be enabled.
MDS.CleanMDReqIDOnSessionDown	boolean Default: True	Clean Market Request IDs on session down.
FIX.TagsToSupress	sequence Defaults: "35 52 34 8 "	Provide a list of tags that needs to be removed from the payload of events emitted from the correlator in the upstream direction. .
MultiContextSupport	boolean	To support multi-context. By enabling this parameter, FIX Server handles all requests in multi-context.

Example com.apama.fix.MDServerConfiguration event:

```
com.apama.fix.MDServerConfiguration("MarketDataSimulator",
    {"RequestKeyParams":"* 336 625"})
```

Supported features

The FIX_MarketDataServer supports the following features:

- Client authentication
- Tick subscription and unsubscription
- Depth subscription and unsubscription
  - Single snapshot
  - Snapshot + updates (full and incremental)
- MarketData request reject by application
- Configurable subscription key construction parameters

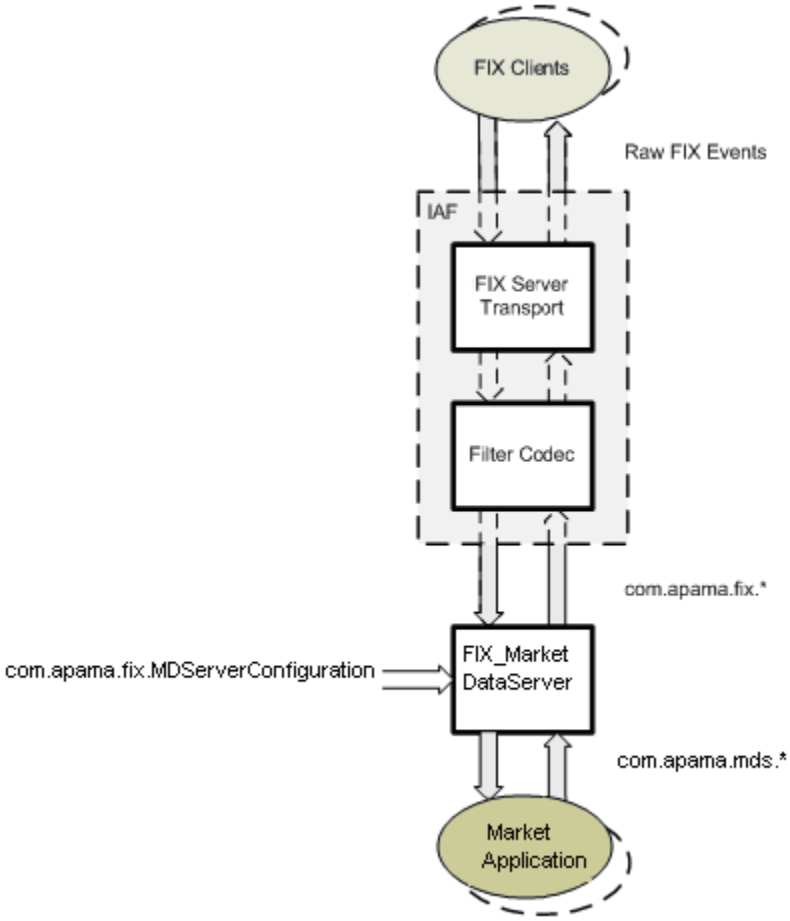
For more information about authentication related configurations, see"Client authentication" on page 28.

MarketData Server uses `com.apama.fix.*` events to interact with the server transport and `com.apama.mds.*` events from MarketDataServerSupport.mon to interact with an application.

When the server receives a `com.apama.fix.MarketDataRequest` event from its transport, it creates a `com.apama.mds.SubscribeDepth/ com.apama.mds.SubscribeTick` event with a unique `MDReqID` for the specified `TargetService`.

For `each com.apama.mds.MDServerDepth` or `com.apama.mds.MDServerTick` event received from an application for known subscriptions, the server generates a `com.apama.fix.MarketDataSnapshotFullRefresh` and `com.apama.fix.MarketDataIncrementalRefresh` based on the request type.

FIX adapter as a market data server



Creating contexts at startup and dynamically

By enabling `MultiContextSupport` session parameter, MarketData Server processes all MarketData requests in multi context mode. By using multi contexts, the performance improvements can be seen in terms of throughput and decreased latency because the wait time will be reduced.

Context mapped to symbol can be done at startup or dynamically. At Startup, only user(BA) can map symbol to context by com.apama.mds.MDServerContextRequest event. If the symbol is already mapped to one context, then MarketData Server will reject that request and acknowledge to user with error message by com.apama.mds.MDServerContextResponse event.

If Symbol is not mapped to any context while handling Subscriptions, then MarketData Server sends a request to user(BA) for create a new Context by com.apama.mds.CreateContextRequest event and expect response by com.apama.mds.CreateContextResponse event. If the user doesn't respond to request, then the subscription will handle in the main context only.

Quote server configuration

The FIX_QuoteServer translates incoming fix quote request messages into its Apama equivalents like `com.apama.mds.SubscribeDepth` events.

Quote Server can be configured and instantiate by sending the following event:

```
com.apama.fix.QServerConfiguration {
        string connection;
        dictionary<string,string> configuration;
    }
```

Configuration parameters

The following session configuration parameters can be used to configure the quote server.

Name	Type	Description
`RequestKeyParams`	`sequence` Default: * (include all `extraParams`	Specifies the parameters or tags that should be considered for creating the subscription key for Depth/Tick subscriptions. The value is specified as a string with elements separated by spaces. ■ An asterisk (`*`) includes all `extraParams` of `MarketDataRequest`. ■ An empty string ("") includes none of the `extraParams` of `MarketDataRequest`. ■ "336 625" includes tags 336, 625 and their values only. ■ "* 336 625" includes all except tags 336 and 625.
`FixChannel`	`String` Default: FIX	The channel to emit upstream events to. See "Session configuration" on page 38.
`TargetService`	`String` Default: MarketSimulator	The ID of the target service.

Name	Type	Description
FIX.TagsToSupress	sequence Defaults: "35 52 34 8 "	Provide a list of tags that needs to be removed from the payload of events emitted from the correlator in the upstream direction. .
MultiContextSupport	boolean	To support multi-context. By enabling this parameter, FIX Server handles all requests in multi-context.
UseSharedSessionManager	boolean Default: False	If vendor is expecting both Quotes and Orders in a single session, you can share Quote session with Order using this parameter.

Example:

```
com.apama.fix.QServerConfiguration("QuoteSimulator",
 {"RequestKeyParams":"* 336 625"})
```

Supported features

- Client authentication
- Quote Request
- Quote Cancel
- Quote Request Reject by Business Application
- Quote Acknowledgement

Quote Server uses `com.apama.fix.*` events to interact with Server Transport and `com.apama.mds.*` events from `MarketDataServerSupport.mon` to interact with the application.

On receiving a `com.apama.fix.QuoteRequest` event from its transport, the server creates `com.apama.mds.SubscribeDepth` with unique `QuoteReqID` for specified TargetService.

For each `com.apama.mds.MDServerDepth` received from the application for known subscriptions, it will generate `com.apama.fix.Quote`.

Creating contexts at startup and dynamically

By enabling `MultiContextSupport` session parameter, Quote Server processes all Quote requests in multi context mode. By using multi contexts the performance improvements can be seen in terms of throughput and decreased latency because the wait time will be reduced.

Context mapped to symbol can be done at startup or dynamically. At Startup, only user (BA) can map symbol to context by `com.apama.mds.MDServerContextRequest` event. If the symbol is already mapped to one context, then Quote Server will reject that request and acknowledge to user with error message by `com.apama.mds.MDServerContextResponse` event.

If Symbol is not mapped to any context while handling Subscriptions, then Quote Server sends a request to user(BA) for create a new context by `com.apama.mds.CreateContextRequest` event and expect response by `com.apama.mds.CreateContextResponse` event. If the user doesn't respond to request, then the subscription will handle in the main context only.

Drop Copy Session

The Drop Copy service allows users to receive real-time copies of Execution Report and Acknowledgment messages as they are sent over Order entry system sessions.

To enable drop copy session user should inject below monitor.

```
"${APAMA_HOME}/adapters/FIX/monitors/FIX_DropCopyManager.mon".
```

To initiate the session, you must configure with below event

```
com.apama.fix.DropCopyConfiguration(<Transport_Name>,
{"SERVICEID":<value>,"FixChannel":<value>})
```

Sample configuration

```
com.apama.fix.DropCopyConfiguration
("connection", {"SERVICEID":"DROP-COPY","FixChannel":"FIX_DROPCOPY"})
```

10 Supported FIX Features

The following is a list of the FIX features which are currently supported by the FIX adapter:

Feature	Supported	Notes
`Heartbeat`	Y	Supported in transport/ QuickFIX
`Logon`	Y	Supported in transport/ QuickFIX
`Test Request`	Y	Supported in transport/ QuickFIX
`Resend Request`	Y	Supported in transport/ QuickFIX
`Reject (session-level)`	Y	Supported in transport/ QuickFIX
`Sequence Reset (Gap Fill)`	Y	Supported in transport/ QuickFIX
`Logout`	Y	Supported in transport/ QuickFIX
`Advertisements`	N	
`Indications of Interest`	N	
`News`	N	
`Email`	N	
`Quote Request`	Y	Restricted (see "FIX_SubscriptionManager" on page 40)

Feature	Supported	Notes
Quote	Y	Restricted (see "FIX_SubscriptionManager" on page 40)
Mass Quote	N	
Quote Cancel	Y	Restricted (see "FIX_SubscriptionManager" on page 40)
Quote Status Request	N	
Quote Acknowledgement	N	
Market Data Request	Y	
Market Data Snapshot / Full Refresh	Y	
Market Data Incremental Refresh	Y	
Market Data Request Reject	Y	
Security Definition Request	Y	Restricted (see "FIX_DataManager" on page 65)
Security Definition	Y	Restricted (see "FIX_DataManager" on page 65)
Security Status Request	N	
Security Status	N	
Trading Session Status Request	Y	
Trading Session Status	Y	
New Order Single	Y	

Feature	Supported	Notes
New Order List	N	
New Order Cross	N	
New Order Multileg	N	
Execution Reports	Y	
Don't Know Trade (DK)	N	
Order Cancel/Replace Request	Y	
Order Cancel Request	Y	
Order Cancel Reject	Y	
Order Status Request	N	
Allocation	N	
Allocation ACK	N	
Settlement Instructions	N	
Bid Request	N	
Bid Response	N	
List Strike Price	N	
List Status	N	
List Execute	N	
List Cancel Request	N	
List Status Request	N	
Business Message Reject	Y	

11 Preparation Checklist

Before attempting to establish a session with a FIX server/client using the FIX adapter, it is essential that at least the following parameters are configured. Depending on the target exchange and application requirements, further configuration may or may not be required in addition.

Configure transport(s)

When acting as client, a separate transport must be configured for each target FIX server to be connected to. The following parameters must be specified for each transport:

Name	Description
`Transport Name`	A name for the transport/session.
`SocketConnectHost` (client only)	The target IP/hostname. (obtained from exchange)
`SocketConnectPort`	The target port. (obtained from the exchange)
`TargetCompID`	Identifies the receiving system. (obtained from the exchange)
`SenderCompID`	Identifies the sending system. (obtained from the exchange)
`DataDictionary`	The path to the data dictionary file. (default: *`adapters`*`/config/FIX42.xml`)

Configure session(s)

Each transport must be identified and configured for the service monitors. Clients transports are configured using a `com.apama.fix.SessionConfiguration` event and server transports are configured using a `com.apama.fix.ServerConfiguration` or `com.apama.fix.MDServerConfiguration` event. The configuration parameters used will depend upon the target system and application requirements.

It is usual practice to create an `*.evt` file containing these events and have them sent in once the service monitors have been injected.

12 Troubleshooting

Although the FIX adapters has a large number of configuration options it is relatively simple to set up and manage as in many cases it is sufficient to use the default configuration files that distribute with the adapter and simply modify the key parameters to suit your environment (host, port, compIds etc). However problems can and do arise and the adapter produces a large amount of diagnostic information to help in resolving such issues.

FIX log files

The FIX adapter produces a number of log files and it is essential to be aware of these when diagnosing problems. Essentially there are three main logs to consider:

The service log

To configure the logLevel and logFile details, see "Setting logging attributes for packages, monitors and events" in *Deploying and Managing Apama Applications* in the Apama documentation.

For example, to change the verbosity of the FIX adapter to DEBUG:

```
engine_management --setApplicationLogLevel com.apama.fix=DEBUG
```

The IAF log

Contains all logging output from the transport(s) and codec as well as the IAF as a whole. The log will be sent to the IAFs `stdout` unless configured via the log switch (`-f`).

The FIX logs

For each FIX session (i.e. transport) the embedded QuickFIX engine will produce a set of log files under the directory that is specified in the 'QuickFIX.FileLogPath' transport property. By default, this is a directory called `FIX_Logs` and is created under the IAF working directory. Usually there will be six files for each session although from a debugging point of view we are primarily interested in:

- `FIX.4.2-<senderCompID>-<targetCompID>.messages.current` – Contains a raw copy of each FIX message that is sent or received by the session.
- `FIX.4.2-<senderCompID>-<targetCompID>.seqnum` – Contains two integers separated by a colon, the first of which is our sequence number and the second of which is the server's sequence number.

When investigating the FIX message log, it is helpful to know the meanings of the various tags. A useful resource for this is FIXionary (http://www.fixionary.com) which has a full listing of all messages within FIX and their corresponding tags.

Diagnosing connectivity/session problems

The first place to look when diagnosing connectivity problems is the IAF log. Each FIX session should produce the following series of messages upon successfully connecting and logging on to the remote system:

```
…
2007-11-30 10:56:12.070 INFO  [2756] -
   FIX.4.2-<Sender>-<Target>::Connecting to
       <host> on port <port>
2007-11-30 10:56:12.480 INFO  [2756] -
   FIX.4.2-<Sender>-<Target>::Initiated logon
       request
2007-11-30 10:56:13.462 INFO  [2756] -
   FIX.4.2-<Sender>-<Target>::Received logon
       response
…
```

If after attempting to connect you do not see “Initiated logon request”, it is likely that a connection to the specified host/port cannot even be established. In this instance you must investigate your environment/network and establish that a route to the target system is actually possible.

If you do see “Initiated logon request” but no response is received or the connection is terminated by the peer in some way then it is likely that the target system has rejected the logon attempt. In this case you need to look at the FIX message log and investigate the message exchange. It may be that you have provided incorrect details or some essential tag is missing from the logon message.

Another possibility in this instance is that there is a sequence number mismatch. The target system will report this to you and tell you what it expects the sequence number to be. In this case you must shutdown the adapter, edit the sequence number file (discussed above) and make sure that the sequence numbers are correctly aligned with what you and the target system are expecting before restarting.

Once a FIX session is properly established and as long as the sessions have been properly configured by sending in the relevant SessionConfiguration event(s) (See "FIX Client Monitors" on page 37, the session manager will notify all the other FIX service monitors that the session(s) are now up. This will also be logged in the service log as:

```
…
[2007-11-30 10:56:13.462 INFO] FIX Session Manager
   [<CONNECTION_NAME>]: Session
         <CONNECTION_NAME>'s IAF is connected
[2007-11-30 10:56:13.462 INFO] FIX Session Manager
   [<CONNECTION_NAME>]: Session
         <CONNECTION_NAME> has been logged on
```

…

The first message tells us that the session manager is receiving heartbeats from the IAF and the second message tells us that the session manager has received a logon message for that connection.

Diagnosing application problems

As mentioned, all FIX service monitors will output diagnostic information to the service logs. This information can be useful in diagnosing application related problems such as missing or incorrectly specified parameters.

For example, upon receiving a new market data subscription request the subscription manager will output information regarding its current reference counts for that symbol and what (if anything) it intends to send to the market.

However, sometimes it is necessary to dig deeper and establish exactly what is being sent to and received from the target system. In these cases it is best to refer to the FIX message log (as discussed above) to diagnose the problem.

Creating a support case

If you encounter a problem that you cannot solve and need to contact support, there are a number of guidelines (in addition to those set out in the preface) that should be followed to ensure your case is resolved as quickly as possible.

Firstly a full description of the problem and the conditions that resulted in its discovery along with any other information that may help in resolving the issue (Such as order ids, subscription information etc). Secondly the following files should be provided:

- IAF configuration file
- IAF Log file
- All FIX logs
- Service Log
- Correlator Replay log

13 Apama Currenex FIX Adapter

IAF adapter configuration

Currenex uses a custom data dictionary FIX42-CNX.xml. This file must be referenced by the `QuickFIX.DataDictionary` transport property. For example,

```
<property name="QuickFIX.DataDictionary"
 value="<APAMA_HOME>/adapters/config/FIX42-CNX.xml"/>.
```

In addition, the following properties are required for Currenex connections:

```
<!-- Password -->
<property name="StaticField.body.A.554" value="<CURRENEX_PASSWORD>"/>
```

Where `CURRENEX_PASSWORD` is the password supplied to you by Currenex.

The supplied sample configuration file FIX-CNX-Legacy.xml.dist contains placeholders for the necessary configuration parameters. This configuration file defines two connections (CURRENEX_TRADING and CURRENEX_MARKET_DATA) using the event channel "CURRENEX_FIX" for communication with the correlator.

Session configuration

Currenex sessions must be configured with the following configuration parameters:

```
Trading sessions:
OrderManager.RequireSessionsStatusOpen = true
OrderManager.IgnoreStatusOnOrderCancelReject = true
OrderManager.HandleSuspendedExecType = true
OrderManager.UseCurrencyFromSymbol = "BASE" or "TERM"
   (Refer to README-FIX.txt for more about this parameter)
OrderManager.CustomOrdTypes = "TRAILING STOP:V,OCO:W,IFD OCO:Y"
This is used to set order types specific to currenex
Market data sessions:
SubscriptionManager.RequireSessionStatusOpen = true
SubscriptionManager.FieldOmissions = Currency
```

Sample session configuration events, compatible with the distributed configuration file:

```
// Currenex trading session
com.apama.fix.SessionConfiguration("CURRENEX_TRADING",
    {"FixChannel":"CURRENEX_FIX", "OrderManager.RequireSessionStatusOpen":"true",
       "OrderManager.IgnoreStatusOnOrderCancelReject":"true",
       "OrderManager.HandleSuspendedExecType":"true",
       "OrderManager.UseCurrencyFromSymbol":"BASE",
       "OrderManager.CustomOrdTypes":"TRAILING STOP:V,OCO:W,IFD OCO:Y"})
// Currenex market data session
com.apama.fix.SessionConfiguration("CURRENEX_MARKET_DATA",
    {"FixChannel":"CURRENEX_FIX",
     "SubscriptionManager.RequireSessionStatusOpen":"true",
     "SubscriptionManager.FieldOmissions":"Currency"})
```

Monitor injection order

The following monitors must be injected into the correlator (in the following order) to support the adapter:

```
monitors/StatusSupport.mon
adapters/monitors/FIX_Events.mon
adapters/monitors/FIX_SessionManager.mon
adapters/monitors/FIX_DataManager.mon
adapters/monitors/FIX_SubscriptionManager.mon
adapters/monitors/FIX_OrderManager.mon
adapters/monitors/FIX_StatusManager.mon
adapters/monitors/FIX_EventViewer.mon
adapters/monitors/FIX_CNX_Support.mon
```

The following CDPs must be injected into the correlator to support the adapter:

```
ASB/projects/Legacy Finance Support/Legacy Finance Support.cdp
```

Password management

The FIX_CNX_Support.mon service monitor can be used to reset the Currenex session password. This monitor file must be injected after all of the standard service monitors, but before any Currenex sessions are configured.

To reset the password for a Currenex session, send the following event to the correlator:

```
com.apama.fix.cnx.PasswordChangeRequest
(<transport>, <requestId>, <userId>, <oldPassword>, <newPassword>)
```

Where,

- <transport> is the session/transport name (for example, "CURRENEX_TRADING")
- <requestId> is a user-supplied key to identify replies to this request
- <userId> is the SenderCompId for this session
- <oldPassword> is the existing password for the session
- <newPassword> is the desired new password for the session

Each request should result in a `com.apama.fix.cnx.PasswordChangeResponse()` event being routed and logged by the monitor:

```
com.apama.fix.cnx.PasswordChangeResponse
(<transport>, <requestId>, <userId>, <success>, <status>, <message>)
```

Where,

- <transport> is the session/transport name
- <requestId> is the user-supplied identifier from the request
- <userId> is the SenderCompId from the request
- <success> is a boolean value indicating whether the password was changed

- <status> is one of the RESPONSE_* constants listed in the FIX_CNX_Support monitor. Typical values are RESPONSE_PASSWORD_CHANGED ("5") or RESPONSE_USER_NOT_RECOGNISED ("3") which Currenex uses as a catch-all failure code.
- <message> is any error message returned by the server

A password reset can be performed at any time after the session has been successfully logged on and a Trading Session Status of "Open" has been sent (this will be logged by the FIX Session Manager). A password reset is required if the server sends a Trading Session Status of "Halted" immediately after a successful logon (this will be logged at WARN level by the FIX CNX Password Manager). Note that you cannot successfully log on with an expired password, so perform a password reset as soon as the "Halted" session status is seen. The session will be locked after too many login attempts with an expired password. The session can be unlocked only by calling Currenex technical support.

Marketdata subscriptions

Examples:

```
//market data request (full book non-aggregated)
com.apama.marketdata.SubscribeDepth
("FIX", "CURRENEX_MARKET_DATA", "EUR/USD",{"264":"0","266":"N","7560":"Y"})
//market data request (full book aggregated)
com.apama.marketdata.SubscribeDepth
("FIX", "CURRENEX_MARKET_DATA", "EUR/USD",{"264":"0","266":"Y"})
//market data request (top of book)
com.apama.marketdata.SubscribeDepth
("FIX", "CURRENEX_MARKET_DATA", "EUR/USD",{"264":"1"})
//Tick request
com.apama.marketdata.SubscribeTick
("FIX", "CURRENEX_MARKET_DATA", "EUR/USD",{"263":"T","264":"1"})
```

Order management

Examples

```
//Limit: Buy Base
com.apama.oms.NewOrder("Currenex:1","EUR/USD",1.2899,"BUY","FOREX LIMIT",
 40000,"FIX","","","CURRENEX_TRADING","","",
  {"15":"EUR","460":"4","21":"1","60":"20141002-00:16:40","59":"1","1":"xxx"})
//Market: Buy Base
com.apama.oms.NewOrder("Currenex:3","GBP/USD",0.0,"BUY","FOREX MARKET",
 20000,"FIX","","","CURRENEX_TRADING","","",
  {"15":"GBP","460":"4","21":"1","60":"20141002-00:16:40","59":"1","1":"progress"})
//Stop Loss: Sell Terms - Stop Side (Bid)
com.apama.oms.NewOrder("Currenex:17","EUR/USD",0.0,"SELL","STOP MARKET",
 5000000,"FIX","","","CURRENEX_TRADING","","",
  {"15":"USD","7534":"1","99":"1.2653","460":"4","59":"1"})
//Stop Limit: Buy Base - Stop Side (Bid)
com.apama.oms.NewOrder("Currenex:17","EUR/USD",1.2670,"BUY","STOP LIMIT",
 1000000,"FIX","","","CURRENEX_TRADING","","",
  {"15":"EUR","7534":"1","99":"1.2650","460":"4","59":"1"})
```

```
//Trailing Stop Sell - Trail the Bid by 2 pips
com.apama.oms.NewOrder("Currenex:17","EUR/JPY",0.0,"SELL","TRAILING STOP",
 30000,"FIX","","","CURRENEX_TRADING","","",
  {"15":"EUR","7534":"1","7587":"0.02","460":"4","59":"1"})
//OCO: Leg 1 is Limit; Leg 2 is Stop
com.apama.oms.NewOrder("Currenex:26","EUR/USD",0.0,"BUY","LIMIT IF TOUCHED",
 3000000,"FIX","","","CURRENEX_TRADING","","",
  {"15":"EUR","7534":"1","59":"0","167":"FOR","460":"4","1":"xxx","7540":"1.0482",
   "7541":"3","7542":"1.0482","7543":"1"})
//IFDOCO order for 2000000 EUR/USD
com.apama.oms.NewOrder("Currenex:32","EUR/USD",0.0,"BUY","BEST LIMIT IF TOUCHED",2000000,
 "FIX","","","CURRENEX_TRADING","","",
  {"15":"EUR","59":"0","167":"FOR","460":"4","1":"progress","7535":"3","7536":"2",
    "7537":"1.10382","7539":"1","7569":"3","7570":"1.10382"})
//ICEBERG Order with hidden quantity. Tag 210 (MaxShow)
indicates quantity to be displayed. The Quantity field of NewOrder holds
the total quantity including the hidden amount.
com.apama.oms.NewOrder("Currenex:17", "EUR/USD",1.27378,"BUY","FOREX ICEBERG",500000,
 "FIX","","", "CURRENEX_TRADING", "", "",
   {"15":"EUR","59":"0","210":"200000","460":"4"})
```

Amending an order

```
//Limit order rate is modified
com.apama.oms.NewOrder("Currenex:11","EUR/USD",1.0378,"BUY","FOREX LIMIT",
 500000,"FIX","","","CURRENEX_TRADING","","",
  {"15":"EUR","460":"4","21":"1","60":"20141002-00:16:40","59":"1","1":"xxx"})
com.apama.oms.AmendOrder("Currenex:11","FIX","EUR/USD",1.27374,"BUY","FOREX LIMIT",
 500000,{"15":"EUR","460":"4","21":"1","60":"20141002-00:16:40","59":"1","1":"xxx"})
```

Cancelling an order

```
com.apama.oms.NewOrder("Currenex:12","EUR/USD",1.0378,"BUY","FOREX LIMIT",
 400000,"FIX","","","CURRENEX_TRADING","","",
  {"15":"EUR","460":"4","21":"1","60":"20141002-00:16:40","59":"1","1":"progress"})
com.apama.oms.CancelOrder("Currenex:12","FIX",
  {"15":"EUR","460":"4","21":"1","60":"20141002-00:16:40","59":"1","1":"progress"})
```

Currenex FIX adapter over Connectivity plug-in

The Currenex connectivity chain uses new connectivity plug-in architecture to connect to Currenex-FIX compliant systems. For more information on connectivity plug-in, see Apama documentation.

Connectivity configuration

The YAML configuration file FIX-CNX-Connectivity.yaml describes the chain and its components. It contains place holders for the necessary configuration parameters to connect Currenex ESP Service.

Connection-related configuration are specified in the `fixSession` stanza on the "FixConnectivityTransport" instance. It requires custom data dictionary `FIX44-CNX.xml`. This file must be referenced by the element `DataDictionary` of `fixSession`. You must update the `fixSession` stanza with relevant values.

Service monitor injection order

The following CDP's and monitors need to be injected into the correlator to support the adapter:

- ${APAMA_HOME}/monitors/StatusSupport.mon
- ${APAMA_HOME}/monitors/ConnectivityPluginsControl.mon
- ${APAMA_HOME}/monitors/ConnectivityPlugins.mon
- ${APAMA_FOUNDATION_HOME}ASB/projects/general_infrastructure/Foundation Utilities/Foundation Utilities.cdp
- ${APAMA_FOUNDATION_HOME}/projects/general_infrastructure/Service Framework/Service Framework.cdp
- ${APAMA_FOUNDATION_HOME}/ASB/projects/general_infrastructure/Data Views/Data View.cdp
- ${APAMA_FOUNDATION_HOME}/ASB/projects/general_infrastructure/General Utils/Utils.cdp
- ${APAMA_FOUNDATION_HOME}/ASB/projects/general_infrastructure/Status Utils/Status Utils_eventdefs-objects.cdp
- ${APAMA_FOUNDATION_HOME}/ASB/projects/Session Manager/Session Manager_eventdefs-classes.cdp
- ${APAMA_FOUNDATION_HOME}/ASB/projects/New Market Data API/New Market Data API_events-classes.cdp
- ${APAMA_FOUNDATION_HOME}/ASB/projects/New Market Data API/New Market Data API_monitors.cdp
- ${APAMA_FOUNDATION_HOME}/ASB/projects/general_infrastructure/Status Utils/Status Utils_monitors.cdp
- ${APAMA_FOUNDATION_HOME}/ASB/projects/Session Manager/Session Manager_monitors.cdp

`APAMA_HOME` is the Apama installation directory.

`APAMA_FOUNDATION_HOME` is the CMF installation directory.

Subscription management

Currenex FIX Adapter uses the CMF MDA MBP (com.apama.md.D) and BBA (com.apama.md.BBA) streams to publish marketData, and supports SPOT subscription only.

Note: Currenex enforces a constraint on the client to send only one subscription request per symbol. To get both BBA and Depth data for same symbol, you can subscribe to Depth stream and use CMF API to get BBA updates.