Apama FIX Adapter

Version 9.10

April 2016

APAMA

This document applies to Apama FIX Adapter Version 9.10 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-910-20160415

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 Overview

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.

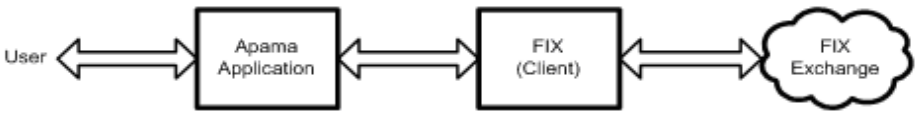
The adapter includes the following components:

- 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.

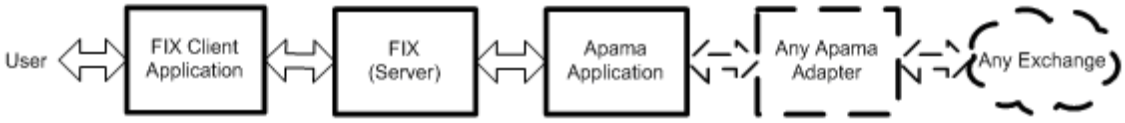
This allows the creation of event based trading applications using the Apama platform. "Figure 1" on page 9 shows a high level diagram of this process:

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 Migrating to FIX adapter 9.10

The following topics provide information for migrating to FIX adapter 9.10.

Migrating from FIX adapter 9.9, 5.3, 5.2, and 5.1

There are no special tasks you need to accomplish to migrate from using FIX adapter 9.9, 5.3, 5.2, and 5.1 to using FIX adapter 9.10.

Migrating from FIX adapter 5.0

If you have an existing Apama 5.0 configuration of the Apama FIX adapter, you can update the Apama installation and the Apama FIX adapter installation, and then perform the following steps to migrate the previous adapter to the current one.

Note: These configuration changes are also required when you are migrating to the version 9.10 FIX Server adapter.

To migrate the FIX adapter

1. The name of the 9.10 Transport library should be changed to `FIXTransport`, so your 5.0 adapter XML configuration files must be updated accordingly. For example:

   ```
   <transport name="ABC_MARKET_DATA" library="FIXTransport">
   ```

2. The static configuration files that are shipped with the adapter have been updated, as follows:

FIX adapter 5.0 file	FIX adapter 9.10 file
`FixNmda-static-codecs.xml`	`FIX-static-codecs.xml`
`FixNmda-static-common.xml`	`FIX-static-common.xml`
`FixNmda-static-groups.xml`	`FIX-static-groups.xml`

The migration from the 5.0 FIX adapter to 9.10 is complete.

Migrating from FIX adapter 4.3

If you have an existing Apama 4.3 configuration of the Apama FIX adapter, you can update the Apama installation and the Apama FIX adapter installation, and then perform the following steps to migrate the previous adapter to the current one.

Note: These configuration changes are also required when you are migrating to the version 9.10 FIX Server adapter.

To migrate the FIX adapter, if you are using custom event definitions, the transport name field provided in the event definitions must be updated. This change is required because of the FIX adapter's dependency on the MDA and FilterCodec configurations. For example, consider the following 4.3 event definition:

```
<event name="News" encoder="FilterCodec" direction="both"
    copyUnmappedToDictionaryPayload="true" package="com.apama.fix">
 <id-rules>
    <downstream>
       <id fields="35" test="==" value="B"/>
    </downstream>
    <upstream/>
 </id-rules>
 <mapping-rules>
    <map apama="TRANSPORT" type="string" default="" transport="TRANSPORT"/>
    <map apama="SESSION" type="string" default="" transport="SESSION"/>
    <map apama="Headline" type="string" default="" transport="148"/>
    <map apama="LinesOfText" type="reference"
       referencetype="sequence<News_LinesOfText>" default="[]" transport="33"/>
    <map type="string" default="B" transport="35"/>
 </mapping-rules>
</event>
```

Revise this event definition for 9.10 to read as follows:

```
<event name="News" encoder="FilterCodec" direction="both"
    copyUnmappedToDictionaryPayload="true" package="com.apama.fix">
 <id-rules>
    <downstream>
       <id fields="35" test="==" value="B"/>
    </downstream>
    <upstream/>
 </id-rules>
 <mapping-rules>
    <map apama="TRANSPORT" type="string" default="" transport="transportName"/>
    <map apama="SESSION" type="string" default="" transport="SESSION"/>
    <map apama="Headline" type="string" default="" transport="148"/>
    <map apama="LinesOfText" type="reference"
       referencetype="sequence <News_LinesOfText>" default="[]" transport="33"/>
    <map type="string" default="B" transport="35"/>
 </mapping-rules>
</event>
```

To migrate the FIX adapter if you are not using standard `FIX-static-codecs.xml`, the codec configuration must be updated. Consider the following:

```
<codec name="FilterCodec" library="FilterCodec" recordTimestamps="both"
    logTimestamps="downstream,upstream,roundtrip">
 <property name="TransportFieldName" value="TRANSPORT"/>
 <property name="filter" value=""/>
 <property name="filter.upstream.44" value="0"/>
</codec>
```

Revise this for 9.10 to read as follows:

```
<codec name="FilterCodec" library="FilterCodec" recordTimestamps="both"
    logTimestamps="downstream,upstream,roundtrip">
 <property name="removeTransportField" value="false"/>
 <property name="TransportFieldName" value="transportName"/>
 <property name="filter" value=""/>
 <property name="filter.upstream.44" value="0"/>
</codec>
```

The migration from the 4.3 FIX adapter to 9.10 is complete.

3 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.

Audit logging

The FIX transport contains a configurable audit logging sub-system that can be used to persist a copy of both incoming and outgoing FIX messages to a file. Currently two logging styles are supported – 'generic' which logs the raw FIX messages and 'iLink' which logs in the CME iLink format. The following parameters are recognized:

- Style - Set the format of the audit log. [generic|ilink]
- Filename – The filename to write to [true|false]
- Truncate – Whether to truncate an existing log [true|false]
- ResetSequence – Whether to reset the sequence number on re-opening the log [true|false]

Example:

```
<property name="Audit.Style" value="ilink"/>
<property name="Audit.Filename" value="C:\logs\ilink_audit.log"/>
```

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.

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.

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>
```

4 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.

5 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>
```

6 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 63.

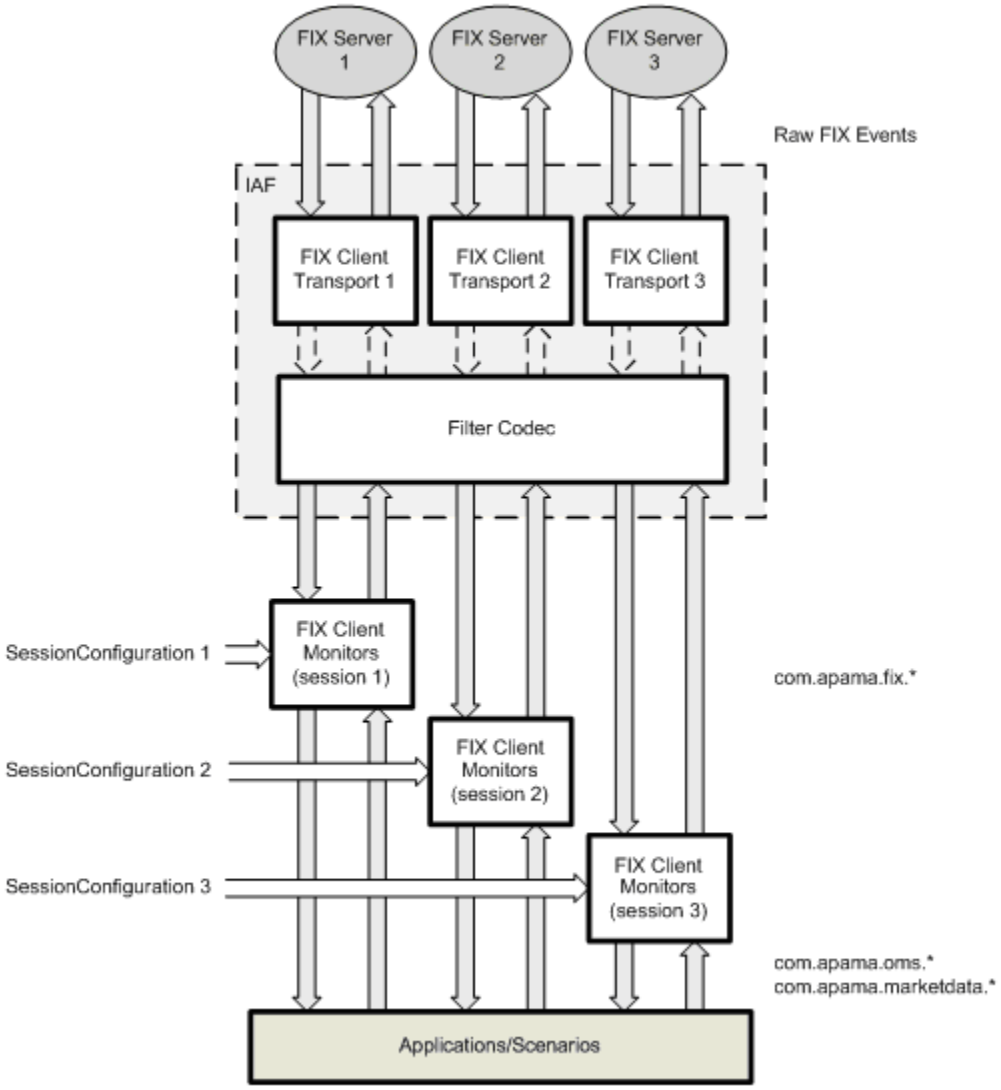
Session configuration

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	Default
`FixChannel`	`String`	The channel to emit upstream events to	"FIX"
`SendSessionStatusRequest`	`Boolean`	Whether to send a FIX `SessionStatusRequest` once logged on.	False

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

Input	Description
`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 21, 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	Default
FixChannel	String	The channel to emit upstream events to	"FIX"
SubscriptionManager. RequireSessionStatusOpen	Boolean	Whether a FIX session status open message must have been received for the session to be active.	False
SubscriptionManager. RequireSessionStatusOpen SubscriptionManager. ResubscribeOnConnect	Boolean	Whether to re-send all subscription requests to the market in the event of a re-connect	False
SubscriptionManager. ResubscribeOnLogon	Boolean	Whether to re-send all subscription requests to the market in the event of a re-logon	True
SubscriptionManager. DistinctDepthTickRequest	Boolean	Whether to send separate depth and tick subscription requests	True
SubscriptionManager. MarketDataFullRefresh	Boolean	Whether to request full refreshes (snapshots) rather	False

Name	Type	Description	Default
		than snapshot + updates	
SubscriptionManager.MarketDataDepthLevel	Integer	Subscription request depth level	0 (Full Book)
SubscriptionManager.FieldOmissions	String	Whitespace delimited set of market data entry fields to omit from the output	""
SubscriptionManager.ForwardMassQuoteRejectReason	boolean	Enable this to forward the QuoteRejectReason from the MassQuoteAcknowledgement message as the reject reason.	False
SubscriptionManager.ClearPricesOnSubscribe	Boolean	Whether to clear all price data prior to issuing a re-subscription.	True
SubscriptionManager.MaxDepthLevels	Integer	Output depth level	0 (Full Book)
SubscriptionManager.SecDefRequestTags	String	Whitespace delimited set of tag numbers to request from the data manager (see "FIX_DataManager" on page 46 for more details)	""
SubscriptionManager.Aggregate	Boolean	Whether to aggregate depth entries with the same side and price.	False
SubscriptionManager.	Boolean	Whether to reuse the request id when	True

Name	Type	Description	Default
ReuseRequestID		re-subscribing or generate a new one each time.	
SubscriptionManager. ReturnZeroPricesOnError	Boolean	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.	True
SubscriptionManager. IncludeTimeInRequestID	Boolean	Specifies that the session startup time should be pre-pended to market data request identifiers.	False
SubscriptionManager. PriceDivisor	Float	Specifies a value that prices in the depth events send to applications will be divided by.	1.0
SubscriptionManager. RecordLatency	Boolean	Adds support for latency measurement on marketdata messages, stringified timestamp set dictionary is added to com.apama.marketdata events.	False
SubscriptionManager. LogLatency	Boolean	Print marketdata latency. Also means that RecordLatency will be enabled.	False

Name	Type	Description	Default
SubscriptionManager.IgnoreSnapshotOnUnsubscription	Boolean	Ignores any trade message received after unsubscription.	True
SubscriptionManager.SupportZeroQuantities	Boolean	Overwrites old quantity if the new quantity is zero	False
SubscriptionManager.MDupdateInPlace	Boolean	Enables the in-place updating of Market data processing (position based updates are applied in-place).	False
SubscriptionManager.SuppressZeroQuantities	Boolean	Suppress zero quantity entries from com.apama.marketdata.Depth event.	False
SubscriptionManager.MDUpdateActionOrder	Sequence	Sequence of MDUpdateAction values separated by spaces. SubscriptionManager specified UpdateAction order will be used to update marketdata from incremental refresh.	"1 2 0" (that is, Change, Delete, New)
SubscriptionManager.PublishNonPositivePrices	Boolean	Publish zero and negative along with positive prices in Depth.	False
SubscriptionManager.SubscriptionKeyTags	String	Specifies that which parameters or tags should be considered for creating the subscription key	""

Name	Type	Description	Default
		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 39.	
`SubscriptionManager. 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 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 40.	""
`SubscriptionManager. RepeatingGroupTags`	`String`	Repeating groups that are part of outgoing request	“”
`SubscriptionManager. UseDefaultTradeEntryTypes`	`Boolean`	Use the default MDEntry type for Trade Subscription	True

Name	Type	Description	Default
SubscriptionManager. UseDefaultDepthEntryTypes	Boolean	Use the default MDEntry type for Depth Subscription	True
SubscriptionManager. AdditionalMDEntryTypes	String	Additional MDEntryTypes to be sent with Subscribe request	“”
SubscriptionManager. RequireSessionStatusOpen	Boolean	Requires Session State to be open to make subscriptions	False
SubscriptionManager. SupportZeroQuantities	Boolean	Overwrites old quantity, if the new quantity is Zero	False
SubscriptionManager. SupportNonUniqueMDEntryIDs	Boolean	Required for exchanges which send same MDEntryID for both BID and Offer Sides	False
SubscriptionManager. QuoteExpireTime	Float	Provides the Quote Expiry Time	60
SubscriptionManager. SubscriptionRequestType	Integer	Provides the Update Mechanism for the subscription (“1” is SNAPSHOT_PLUS_ UPDATES)	“1”
SubscriptionManager. SendQuoteAck	Boolean	Send Quote Acknowledgment	False
SubscriptionManager. delayBeforeResubscription	Float	Time to wait before sending a subscription request	0
SubscriptionManager. UpdateDataWithoutEntryId	Boolean	Custom handling for updating Order	False

Name	Type	Description	Default
		book when EntryID is not present	
SubscriptionManager.AcceptNonPositivePrices	Boolean	Required to support Zero or Negative prices	False
SubscriptionManager.ForwardMDReqID	Boolean	Forward MDReqID used in subscription along with Depth updates	False
SubscriptionManager.UseCurrencyFromSymbol	String	Extract the Currency information from the Symbol. Accepted Values are “BASE” and “TERM”. For more information, see "UseCurrencyFromSymbol note" on page 45.	“BASE”
SubscriptionManager.ValidateDepthSubErrorOnQuoteCancel	Boolean	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.	False
SubscriptionManager.TagsToForwardFromExchange	String	List of tags received from exchange to be forward to the user	“”

Name	Type	Description	Default
SubscriptionManager. ReturnZeroPricesOnError	Boolean	Return empty depth/tick on subscription error	True
SubscriptionManager. EnableMassQuote Acknowledgement	Boolean	Enable Mass Quote Acknowledgment	False
SubscriptionManager. UnsubscribeBeforeSubscribe	Boolean	Send an unsubscribe request to the exchange before making a subscribe request	False
SubscriptionManager. GenerateInstrumentID	Boolean	Lets the adapter generate an instrumentID	False
SubscriptionManager. IncludeTimeInRequestID	Boolean	Include Time in Request ID	False
SubscriptionManager. IgnoreUnexpectedUpdate	Boolean	Ignore Unexpected Market Data Update	False
SubscriptionManager. UnsubscribeOnQuoteCancel	Boolean	Remove subscription when a QuoteCancel is received form exchange	False
SubscriptionManager. IgnoreQuoteCancel OnUnsubscription	Boolean	This parameter will handle the QuoteCancel messages received as the response to the Quote Unsubscription.	False

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.

WARNING! 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
`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	Default
`FixChannel`	`String`	The channel to emit upstream events to	"FIX"
`OrderManager.RequireSessionStatusOpen`	`Boolean`	Whether a FIX session status open message must have been received for the session to be active.	False
`OrderManager.OrderIDServiceName`	`String`	The service to request order ids from	"FIX"
`bustTime`	`String`	Either a decimal number days (relative) or a `cron` format (absolute) time to continue listening for busted trade execution	"" (0 days)

Name	Type	Description	Default
		reports after an order has become final.	
`OrderManager.SecDefRequestTags`	`String`	Whitespace delimited set of tag numbers to request from the data manager (see "FIX_DataManager" on page 46 for more details)	""
`OrderManager.CopyExecTypeToOrdStatus`	`Boolean`	Whether to simply assign the value of ExecType to OrdStatus thereby ensuring they are always identical.	False
`OrderManager.useMillisecondTransactTimes`	`Boolean`	Whether to use millisecond accuracy on order transact times.	False
`OrderManager.IgnoreStatusOnOrderCancelReject`	`Boolean`	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 `origClorderId` set to clorderId of the last successful order.	False
`OrderManager.KillOrdersOnSessionDown`	`Boolean`	Cancels all order listeners when the session goes down.	True
`OrderManager.RecordLatency`	`Boolean`	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,	False

Name	Type	Description	Default
		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`	Print Order management latency, also enables OrderManager.RecordLatency	False
`OrderManager.CancelRejectUsesOrigId`	`Boolean`	Handles Order Cancel Reject with swapped `ClOrdID(11)` and `OrigClOrdID(41)` fields.	False
`OrderManager.amendUsesNonRejectedClOrderID`	`Boolean`	Specifies that order amend/cancel uses last `ClOrderID` if it is not rejected.	False
`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 45.	
`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 for downstream messages). The value of this should be the number of the FIX tag that the alternative	

Name	Type	Description	Default
		security ID will go in. For more information about this parameter, see "UseAltSecurityId note" on page 40.	
FIX.TagsToSupress	Sequence	Provide a list of tags that needs to be removed from the payload of events emitted from the correlator in the upstream direction.	"35 52 34 8"
OrderManager. GenerateNewStyle OrderIds	Boolean	Generates the id with the complete current (full length) time for uniqueness.	FALSE
OrderManager. HandleSuspended ExecType	Boolean	Required to handle Suspended order ExecType	FALSE
OrderManager. CustomOrdTypes	String	Supports custom Order Types	“”
OrderManager. updateKeyParams OnStateChange	Boolean	Price/Qty fields of order update will update only on an amend order is accepted	FALSE
OrderManager. SendExecutionAck	Boolean	Sends back acknowledgment to Execution Report	FALSE
OrderManager. CopyClOrdIDTo OrderUpdate	Boolean	Forwards ClOrderID sent to the exchange in Order Update	FALSE

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.

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	Default
`FixChannel`	`String`	The channel to emit upstream events to	"FIX"

Name	Type	Description	Default
`DataManager. SecurityRequestType`	`String`	The security request type to send to the market (see the FIX specification for more details)	3 (Request List Securities)
`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.	

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"

Input	Description
`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 have been received before being available.	false
`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	""

Name	Type	Description	Default
		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`).

7 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:

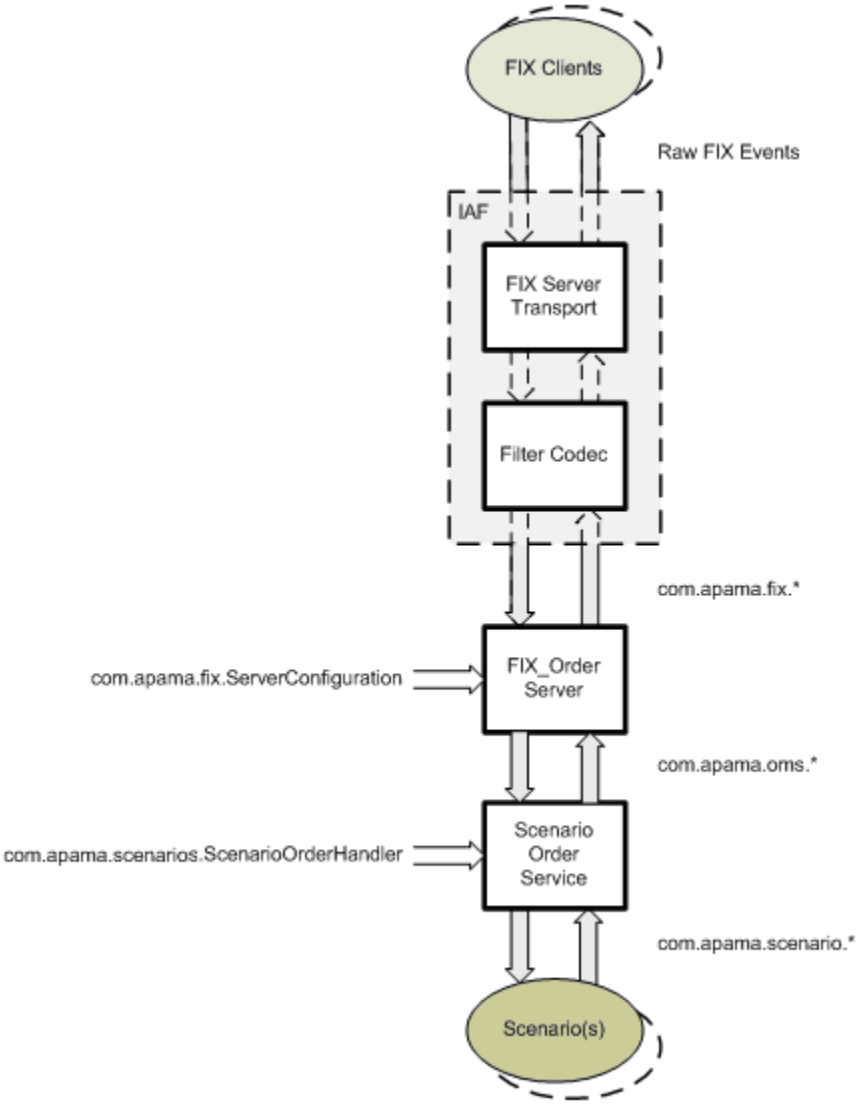
```
com.apama.fix.SessionConfiguration("Connection",{"NMDA_FIX_SESSION_NAME":"FIX"})
```

8 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 57). 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:

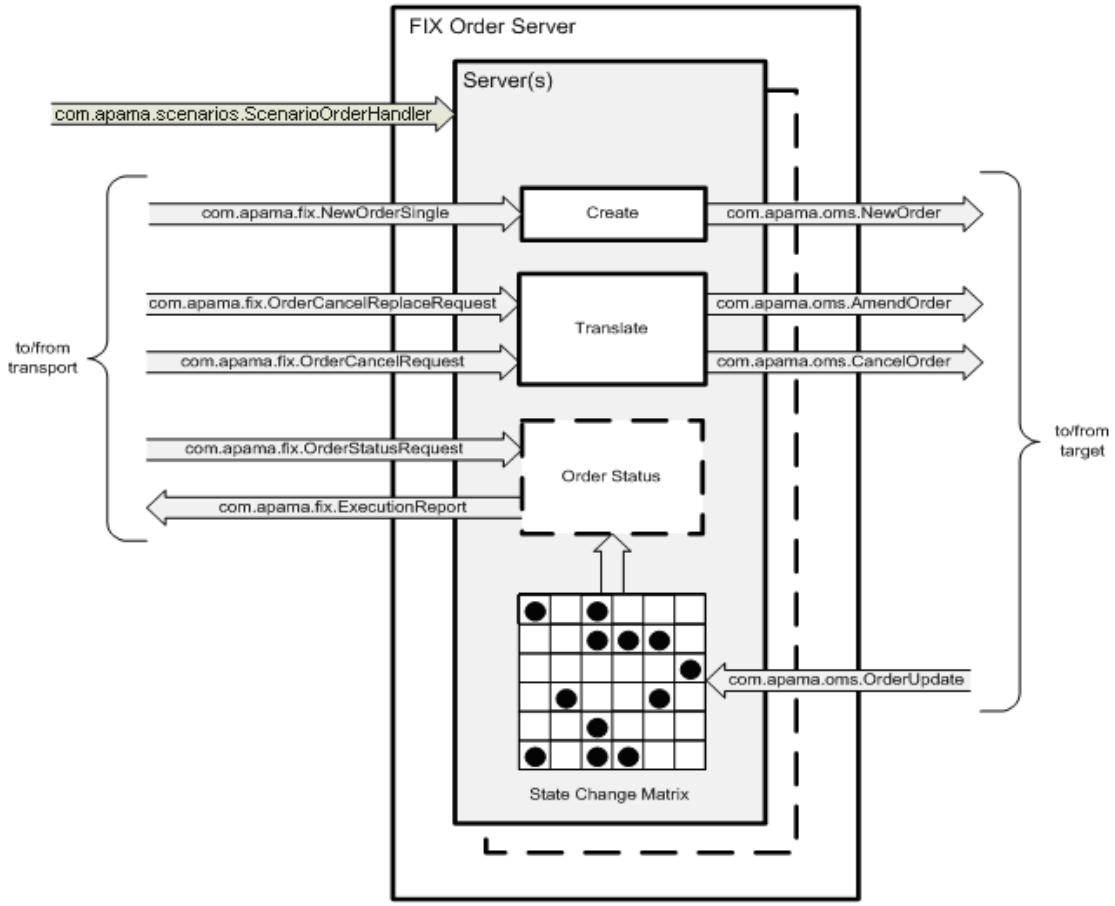
```
com.apama.oms.NewOrder("Server:1e+09:1","ORCL",0,"BUY","MARKET",10000,
   "MarketSimulator","TestBrokerID","","Server","","Client_1_ID",
   {"109":"Client_1_ID","Account":"",
   "__timestamps":"{}","strgy1":"100","strgy2":"200","strgy3":"300"})
```

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	Default
`FixChannel`	`String`	The channel to emit upstream events to.	`FIX`
`TargetService`	`String`	The ID of the target service	`MarketSimulator`
`ScenarioParameter`	`string`	The FIX field to extract the scenario ID from	`TargetStrategy` (847)

ScenarioOrderService

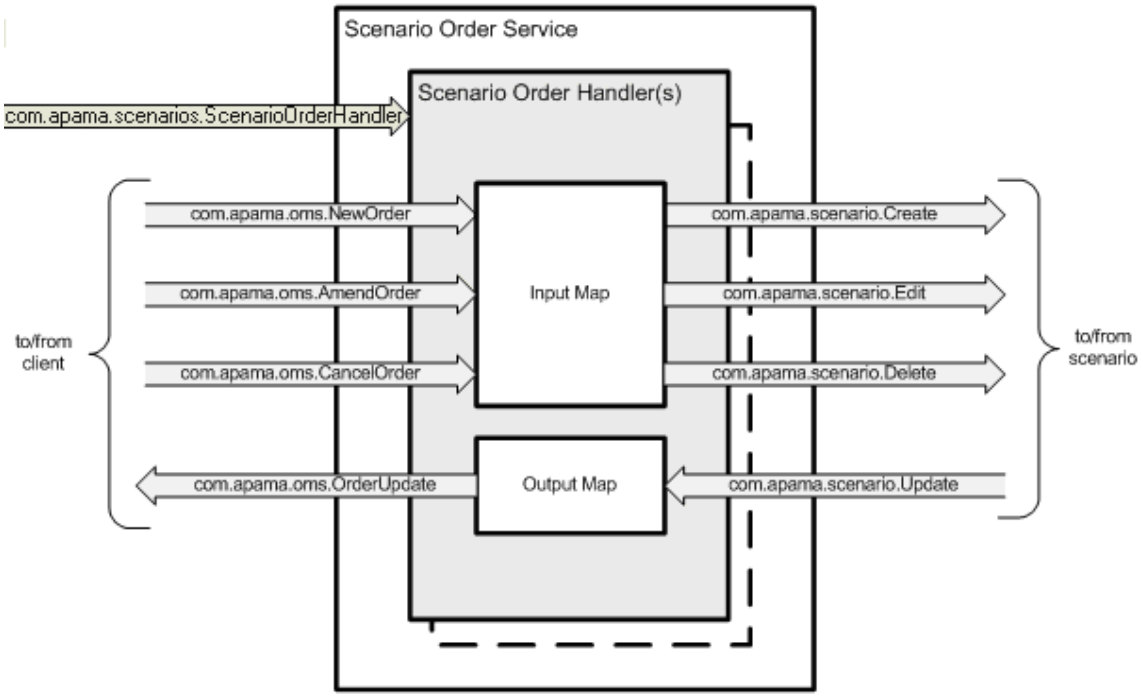
The `ScenarioOrderService` is designed around the notion of a scenario order handler which is an object that translates order events for a particular scenario. The scenario order service can contain many handlers and therefore can manage many scenarios simultaneously. A handler is configured by sending 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 6" on page 58) 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	Default
`RequestKeyParams`	`sequence`	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.	* (include all `extraParams`

Name	Type	Description	Default
		■ 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	The channel to emit upstream events to.	FIX
TargetService	String	The ID of the target service.	MarketSimulator

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 19.

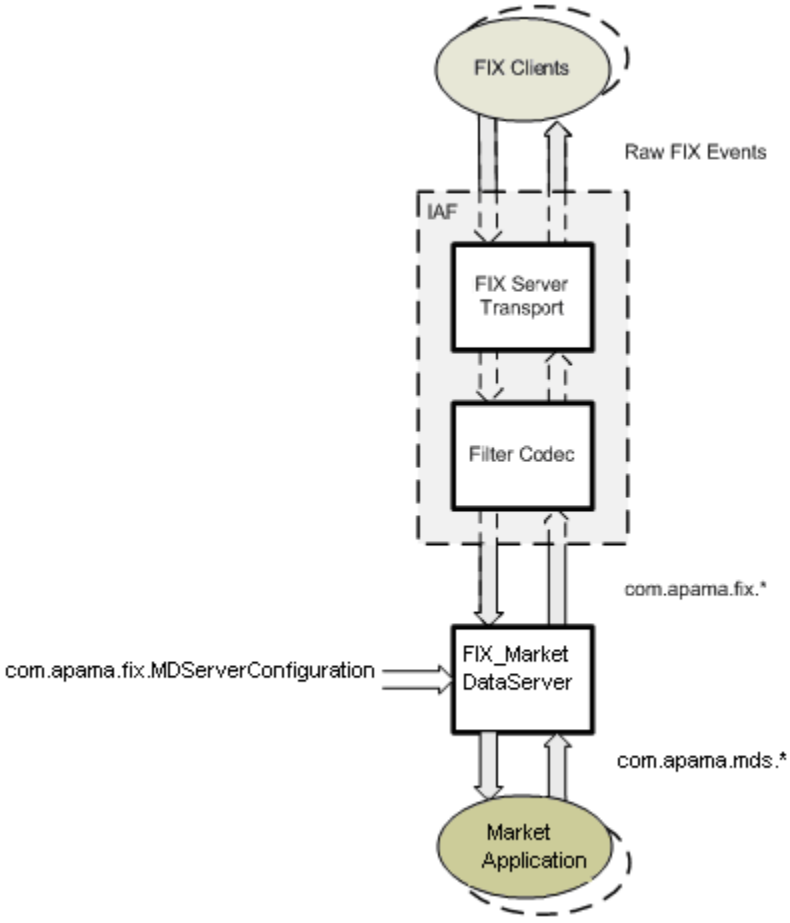
MarketData Server uses com.apama.fix.* events to interact with the server transport and com.apama.mds.* events from MarketDataServerSupport.mon to interact with an application.

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



9 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 30)

Feature	Supported	Notes
Quote	Y	Restricted (see "FIX_SubscriptionManager" on page 30)
Mass Quote	N	
Quote Cancel	Y	Restricted (see "FIX_SubscriptionManager" on page 30)
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 46)
Security Definition	Y	Restricted (see "FIX_DataManager" on page 46)
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	

10 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.

11 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 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 27, 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. The volume and detail of this information is controlled by the log level as set by the SetLogLevel(…) event. 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