Quickfix

QuickFIX/J Component

The **quickfix** component adapts the QuickFIX/J FIX engine for using in Camel . This component uses the standard Financial Interchange (FIX) protocol for message transport.

Previous Versions
The **quickfix** component was rewritten for Camel 2.5. For information about using the **quickfix** component prior to 2.5 see the documentation section below.

Maven users will need to add the following dependency to their `pom.xml` for this component:

xml<dependency> <groupId>org.apache.camel</groupId> <artifactId>camel-quickfix</artifactId> <version>x.x.x</version> <!-- use the same version as your Camel core version --> </dependency>

URI format

quickfix:configFile[?sessionID=sessionID&lazyCreateEngine=true|false]

The **configFile** is the name of the QuickFIX/J configuration to use for the FIX engine (located as a resource found in your classpath). The optional **sessionI** identifies a specific FIX session. The format of the sessionID is:

(BeginString):(SenderCompID)[/(SenderSubID)[/(SenderLocationID)]]->(TargetCompID)[/(TargetSubID)[/(TargetLocationID)]]

The optional **lazyCreateEngine** (Camel 2.12.3+) parameter allows to create QuickFIX/J engine on demand. Value **true** means the engine is started when first message is send or there's consumer configured in route definition. When **false** value is used, the engine is started at the endpoint creation. When this parameter is missing, the value of component's property **lazyCreateEngines** is being used.

Example URIs:

quickfix:config.cfg quickfix:config.cfg?sessionID=FIX.4.2:MyTradingCompany->SomeExchange quickfix:config.cfg?sessionID=FIX.4.2:MyTradingCompany->SomeExchange&lazyCreateEngine=true

Endpoints

FIX sessions are endpoints for the **quickfix** component. An endpoint URI may specify a single session or all sessions managed by a specific QuickFIX/J engine. Typical applications will use only one FIX engine but advanced users may create multiple FIX engines by referencing different configuration files in **quickfix** component endpoint URIs.

When a consumer does not include a session ID in the endpoint URI, it will receive exchanges for all sessions managed by the FIX engine associated with the configuration file specified in the URI. If a producer does not specify a session in the endpoint URI then it must include the session-related fields in the FIX message being sent. If a session is specified in the URI then the component will automatically inject the session-related fields into the FIX message.

Exchange Format

The exchange headers include information to help with exchange filtering, routing and other processing. The following headers are available:

confluenceTableSmall

Header Name	Description
EventCategory	One of `AppMessageReceived`, `AppMessageSent`, `AdminMessageReceived`, `AdminMessageSent`, `SessionCreated`, `SessionLogon`, `SessionLogoff`. See the `QuickfixjEventCategory` enum.
SessionID	The FIX message SessionID
MessageType	The FIX MsgType tag value
DataDictionary	Specifies a data dictionary to used for parsing an incoming message. Can be an instance of a data dictionary or a resource path for a QuickFIX/J data dictionary file

The DataDictionary header is useful if string messages are being received and need to be parsed in a route. QuickFIX/J requires a data dictionary to parse certain types of messages (with repeating groups, for example). By injecting a DataDictionary header in the route after receiving a message string, the FIX engine can properly parse the data.

QuickFIX/J Configuration Extensions

When using QuickFIX/J directly, one typically writes code to create instances of logging adapters, message stores and communication connectors. The **quickfix** component will automatically create instances of these classes based on information in the configuration file. It also provides defaults for many of the common required settings and adds additional capabilities (like the ability to activate JMX support).

The following sections describe how the **quickfix** component processes the QuickFIX/J configuration. For comprehensive information about QuickFIX/J configuration, see the QFJ user manual.

Communication Connectors

When the component detects an initiator or acceptor session setting in the QuickFIX/J configuration file it will automatically create the corresponding initiator and/or acceptor connector. These settings can be in the default or in a specific session section of the configuration file.

confluenceTableSmall

Session Setting	Component Action
`ConnectionType=initiator`	Create an initiator connector
`ConnectionType=acceptor`	Create an acceptor connector

The threading model for the QuickFIX/J session connectors can also be specified. These settings affect all sessions in the configuration file and must be placed in the settings default section.

confluenceTableSmall

Default/Global Setting	Component Action
`ThreadModel=ThreadPerConnector`	Use `SocketInitiator` or `SocketAcceptor` (default)
`ThreadModel=ThreadPerSession`	Use `ThreadedSocketInitiator` or `ThreadedSocketAcceptor`

Logging

The QuickFIX/J logger implementation can be specified by including the following settings in the default section of the configuration file. The `ScreenLog` is the default if none of the following settings are present in the configuration. It's an error to include settings that imply more than one log implementation. The log factory implementation can also be set directly on the Quickfix component. This will override any related values in the QuickFIX/J settings file.

confluenceTableSmall

Default/Global Setting	Component Action
`ScreenLogShowEvents`	Use a `ScreenLog`
`ScreenLogShowIncoming`	Use a `ScreenLog`
`ScreenLogShowOutgoing`	Use a `ScreenLog`
`SLF4J*`	**Camel 2.6+**. Use a `SLF4JLog`. Any of the SLF4J settings will cause this log to be used.
`FileLogPath`	Use a `FileLog`
`JdbcDriver`	Use a `JdbcLog`

Message Store

The QuickFIX/J message store implementation can be specified by including the following settings in the default section of the configuration file. The `MemoryStore` is the default if none of the following settings are present in the configuration. It's an error to include settings that imply more than one message store implementation. The message store factory implementation can also be set directly on the Quickfix component. This will override any related values in the QuickFIX/J settings file.

confluenceTableSmall

Default/Global Setting	Component Action
`JdbcDriver`	Use a `JdbcStore`
`FileStorePath`	Use a `FileStore`
`SleepycatDatabaseDir`	Use a `SleepcatStore`

Message Factory

A message factory is used to construct domain objects from raw FIX messages. The default message factory is `DefaultMessageFactory`. However, advanced applications may require a custom message factory. This can be set on the QuickFIX/J component.

JMX

confluenceTableSmall

Default/Global Setting	Component Action
`UseJmx`	if `Y`, then enable QuickFIX/J JMX

Other Defaults

The component provides some default settings for what are normally required settings in QuickFIX/J configuration files. `SessionStartTime` and `SessionEndTime` default to "00:00:00", meaning the session will not be automatically started and stopped. The `HeartBtInt` (heartbeat interval) defaults to 30 seconds.

Minimal Initiator Configuration Example

[SESSION] ConnectionType=initiator BeginString=FIX.4.4 SenderCompID=YOUR_SENDER TargetCompID=YOUR_TARGET

Using the InOut Message Exchange Pattern

Camel 2.8+

Although the FIX protocol is event-driven and asynchronous, there are specific pairs of messages that represent a request-reply message exchange. To use an InOut exchange pattern, there should be a single request message and single reply message to the request. Examples include an OrderStatusRequest message and UserRequest.

Implementing InOut Exchanges for Consumers

Add "exchangePattern=InOut" to the QuickFIX/J enpoint URI. The `MessageOrderStatusService` in the example below is a bean with a synchronous service method. The method returns the response to the request (an ExecutionReport in this case) which is then sent back to the requestor session.

from("quickfix:examples/inprocess.cfg?sessionID=FIX.4.2:MARKET->TRADER&exchangePattern=InOut") .filter(header(QuickfixjEndpoint.MESSAGE_TYPE_KEY).isEqualTo(MsgType.ORDER_STATUS_REQUEST)) .bean(new MarketOrderStatusService());

Implementing InOut Exchanges for Producers

For producers, sending a message will block until a reply is received or a timeout occurs. There is no standard way to correlate reply messages in FIX. Therefore, a correlation criteria must be defined for each type of InOut exchange. The correlation criteria and timeout can be specified using `Exchange` properties.

Description	Key String	Key Constant	Default
Correlation Criteria	"CorrelationCriteria"	QuickfixjProducer.CORRELATION_CRITERIA_KEY	None
Correlation Timeout in Milliseconds	"CorrelationTimeout"	QuickfixjProducer.CORRELATION_TIMEOUT_KEY	1000

The correlation criteria is defined with a `MessagePredicate` object. The following example will treat a FIX ExecutionReport from the specified session where the transaction type is STATUS and the Order ID matches our request. The session ID should be for the *requestor*, the sender and target CompID fields will be reversed when looking for the reply.

exchange.setProperty(QuickfixjProducer.CORRELATION_CRITERIA_KEY, new MessagePredicate(new SessionID(sessionID), MsgType.EXECUTION_REPORT) .withField(ExecTransType.FIELD, Integer.toString(ExecTransType.STATUS)) .withField(OrderID.FIELD, request.getString(OrderID.FIELD)));

Example

The source code contains an example called `RequestReplyExample` that demonstrates the InOut exchanges for a consumer and producer. This example creates a simple HTTP server endpoint that accepts order status requests. The HTTP request is converted to a FIX OrderStatusRequestMessage, is augmented with a correlation criteria, and is then routed to a quickfix endpoint. The response is then converted to a JSON-formatted string and sent back to the HTTP server endpoint to be provided as the web response.

The Spring configuration have changed from Camel 2.9 onwards. See further below for example.

Spring Configuration

Camel 2.6 - 2.8.x

The QuickFIX/J component includes a Spring `FactoryBean` for configuring the session settings within a Spring context. A type converter for QuickFIX/J session ID strings is also included. The following example shows a simple configuration of an acceptor and initiator session with default settings for both sessions.

{snippet:id=e1|lang=xml|url=camel/branches/camel-2.8.x/components/camel-quickfix/src/test/resources/org/apache/camel/component/quickfixj/QuickfixjSpringTest-context.xml}

Camel 2.9 onwards

The QuickFIX/J component includes a `QuickfixjConfiguration` class for configuring the session settings. A type converter for QuickFIX/J session ID strings is also included. The following example shows a simple configuration of an acceptor and initiator session with default settings for both sessions.

{snippet:id=e1|lang=xml|url=camel/trunk/components/camel-quickfix/src/test/resources/org/apache/camel/component/quickfixj/QuickfixjSpringTest-context.xml}

Exception handling

QuickFIX/J behavior can be modified if certain exceptions are thrown during processing of a message. If a `RejectLogon` exception is thrown while processing an incoming logon administrative message, then the logon will be rejected.

Normally, QuickFIX/J handles the logon process automatically. However, sometimes an outgoing logon message must be modified to include credentials required by a FIX counterparty. If the FIX logon message body is modified when sending a logon message (EventCategory=`AdminMessageSent` the modified message will be sent to the counterparty. It is important that the outgoing logon message is being processed *synchronously*. If it is processed asynchronously (on another thread), the FIX engine will immediately send the unmodified outgoing message when it's callback method returns.

FIX Sequence Number Management

If an application exception is thrown during *synchronous* exchange processing, this will cause QuickFIX/J to not increment incoming FIX message sequence numbers and will cause a resend of the counterparty message. This FIX protocol behavior is primarily intended to handle *transport* errors rather than application errors. There are risks associated with using this mechanism to handle application errors. The primary risk is that the message will repeatedly cause application errors each time it's re-received. A better solution is to persist the incoming message (database, JMS queue) immediately before processing it. This also allows the application to process messages asynchronously without losing messages when errors occur.

Although it's possible to send messages to a FIX session before it's logged on (the messages will be sent at logon time), it is usually a better practice to wait until the session is logged on. This eliminates the required sequence number resynchronization steps at logon. Waiting for session logon can be done by setting up a route that processes the `SessionLogon` event category and signals the application to start sending messages.

See the FIX protocol specifications and the QuickFIX/J documentation for more details about FIX sequence number management.

Route Examples

Several examples are included in the QuickFIX/J component source code (test subdirectories). One of these examples implements a trival trade excecution simulation. The example defines an application component that uses the URI scheme "trade-executor".

The following route receives messages for the trade executor session and passes application messages to the trade executor component.

```
from("quickfix:examples/inprocess.cfg?sessionID=FIX.4.2:MARKET->TRADER"). filter(header(QuickfixjEndpoint.EVENT_CATEGORY_KEY).isEqualTo(QuickfixjEventCategory.AppMessageReceived)). to("trade-executor:market");
```

The trade executor component generates messages that are routed back to the trade session. The session ID must be set in the FIX message itself since no session ID is specified in the endpoint URI.

```
from("trade-executor:market").to("quickfix:examples/inprocess.cfg");
```

The trader session consumes execution report messages from the market and processes them.

```
from("quickfix:examples/inprocess.cfg?sessionID=FIX.4.2:TRADER->MARKET"). filter(header(QuickfixjEndpoint.MESSAGE_TYPE_KEY).isEqualTo(MsgType.EXECUTION_REPORT)). bean(new MyTradeExecutionProcessor());
```

QuickFIX/J Component Prior to Camel 2.5

The **quickfix** component is an implementation of the QuickFIX/J engine for Java . This engine allows to connect to a FIX server which is used to exchange financial messages according to FIX protocol standard.

Note: The component can be used to send/receives messages to a FIX server.

URI format

quickfix-server:config file quickfix-client:config file

Where **config file** is the location (in your classpath) of the quickfix configuration file used to configure the engine at the startup.

Note: Information about parameters available for quickfix can be found on QuickFIX/J web site.

The quickfix-server endpoint must be used to receive from FIX server FIX messages and quickfix-client endpoint in the case that you want to send messages to a FIX gateway.

Exchange data format

The QuickFIX/J engine is like CXF component a messaging bus using MINA as protocol layer to create the socket connection with the FIX engine gateway.

When QuickFIX/J engine receives a message, then it create a QuickFix.Message instance which is next received by the camel endpoint. This object is a 'mapping object' created from a FIX message formatted initially as a collection of key value pairs data. You can use this object or you can use the method 'toString' to retrieve the original FIX message.

Note: Alternatively, you can use camel bindy dataformat to transform the FIX message into your own java POJO

When a message must be send to QuickFix, then you must create a QuickFix.Message instance.

Lazy creating engines

From **Camel 2.12.3** onwards, you can configure the QuickFixComponent to lazy create and start the engines, which then only start these on-demand. For example you can use this when you have multiple Camel applications in a cluster with master/slaves. And want the slaves to be standby.

Samples

Direction : to FIX gateway

xml<route> <from uri="activemq:queue:fix"/> <bean ref="fixService" method="createFixMessage"/> // bean method in charge to transform message into a QuickFix.Message <to uri="quickfix-client:META-INF/quickfix/client.cfg"/> // Quickfix engine who will send the FIX messages to the gateway </route>

Direction : from FIX gateway

xml<route> <from uri="quickfix-server:META-INF/quickfix/server.cfg"/> // QuickFix engine who will receive the message from FIX gateway <bean ref=" fixService" method="parseFixMessage"/> // bean method parsing the QuickFix.Message <to uri="uri="activemq:queue:fix"/>" </route>

Endpoint See Also