FIXEdge

Business Rules Guide

Document History

Version	Date	Description of change
1.0	2007-06-19	Initial version
2.0	2009-05-15	Changed “FIXEdge events” section Added sample with events in “Routing with source identifiers” section

Table of Contents

Overview

FIXEdge offers the set of Business Rules, which can be useful to determine message routing, session events, scripting, message modification, plug-in connection, histories management, automatic routing for some types of the messages. The Rules formation is defined by special syntax in the XML format that is described in this guide.

Configuring Business Rules

Business Rules are stored in '*FixEdge/conf/BL_Config.xml'* file in XML format. The name of configuration file can be changed in '*FixEdge.properties*' file using '*BusinessLayer.RoutingRules'* property. The XML formatting is described in '*FixEdge/conf/BusinessLayer.dtd'*. Rules can be changed from FECC or by editing the file directly. They can also be reloaded immediately from FECC.
The following classes are defined for Business Rules:

- Routing rules – used to transform and route FIX messages to sessions, adaptors etc
- Session events – used to handle events that are launched during execution – create session, destroy session etc

The Business Rules XML file may contain the following auxiliary entities definitions that could be used in Business Rules:

- Plug-ins – objects that encapsulate field transformation algorithm
- Handlers - objects that encapsulate message handling algorithm
- Histories – objects that store data from the handled messages
- Routing strategies - objects that encapsulate algorithms of automatic message routing

Loading XML rules from separate files

XML rules for Business Layer can be stored in separate files. Use the following configuration property:
BusinessLayer.RoutingRules = file1.xml, file2.xml, file3.xml,…etc to load the set of rules.

Please note while configuring: each XML document must be a part of a single document. I.e. the first file must contain the standard header of XML file <?xml version="1.0" encoding="UTF-8"?> (or UTF-16).
The certain rules will be loaded into Business Layer from documents, but within tags:

```
<FixEdge>
        <BusinessLayer>
                <!—-
                the rules set...
                -->
        </BusinessLayer>
</FixEdge>
```

The good practice is to start the list with *BL_header.xml* with the following content:

```
<?xml version="1.0" encoding="UTF-8"?> (or UTF-16).
<FixEdge>
        <BusinessLayer>
```

and end up with *BL_tail.xml* with the following content:

```
        </BusinessLayer>
</FixEdge>
```

In this case the configuration property will look like this:
BusinessLayer.RoutingRules = BL_header.xml,…file1.xml, file2.xml,…,BL_tail.xml

Cases of routing rules

FIXEdge router provides the user with a possibility to establish a set of Business Rules that enables FIXEdge with:

- Routing of incoming and outgoing FIX messages (via the correspondence between ClientID and FIX Session).
- Filtering of incoming and outgoing FIX messages.
- Transformation of incoming and outgoing FIX messages.

All 3 functions are accomplished through the customizable (user-defined) Rules in XML format, placed in the Business Layer configuration file. This means that more than one rule can be applied to a message. It is also possible to route one message to multiple destinations or to the same destination multiple times. All rules are applied separately (independently), the order is not guaranteed. Even if a set of transformation actions is applied to messages, the next rule will be applied to the original message.
More information about routing rules can be found in *FixEdge_RoutingRules_UserGuide.html*.

Routing rules for incoming messages from FIX session

While routing an incoming message the Rule must have <Session> source in its structure. Also, there must be at least one instruction in the Action section. On demand, user can choose and specify the condition.

```
<Rule>
  <Source>
    <FixSession SenderCompID="TestSender" TargetCompID="TestTarget"/>
  </Source>
  <Condition>
    <MatchField Field="35" Value=".*"/>
  </Condition>
   <Action>
    <Send>
       <Client Name="TestClient"/>
    </Send>
  </Action>
</Rule>
```

Routing rules for incoming messages from TA client

The routing approach for handling the incoming messages from TA client is the same as for incoming messages from FIX sessions, except for one difference: the source is always a Client ID. FIX messages from TA clients are routed to the requisite FIX sessions based on the SenderCompID and TargetCompID fields.
Example:

```
<Rule>
  <Source>
    <Client Name="TestClient"/>
  </Source>
  <Condition>
    <MatchField Field="35" Value=".*"/>
    <MatchField Field="571" Value="IST-\\d+"/>
  </Condition>
   <Action>
    <Send>
       <FixSession SenderCompID="TestSender" TargetCompID="TestTarget"/>
    </Send>
  </Action>
</Rule>
```

It is possible to send a message to destinations denoted as the SenderCompID and TargetCompID in the message itself. You can do this by omitting SenderCompID and TargetCompID attributes in FixSession tag. For example: <FixSession/>.

Routing with source identifiers

Source identifiers can be used as “Name” attribute of <Source> tag as well as “Name” attribute of <Send> tag. The following example demonstrates the routing of an incoming message from the session identified as “TestSource” to destination with “TestTarget” identifier; only messages that satisfy <MatchField> conditions are included:

```
<Rule>
  <Source Name = "TestSource"/>
          <Condition>
          <MatchField Field="35" Value="D"/>
          </Condition>
          <Action>
               <Send Name="TestTarget"/>
          </Action>
</Rule>
<--! Send reject on back-way if source “TestTarget” are disconnected -->
<OnUndeliveredMessageEvent>
          <Source Name= "TestTarget" />
          <Action>
                    <CreateReject RejectType="application"/>
                    <Send Name="TestSource" />
          </Action>
</OnUndeliveredMessageEvent>
```

Note that the source identifier represents FIX Session or Clients too. Please refer to Source identification section for a detailed description of its usage.

Routing to multiple destinations

The sending rule is able to send a message to multiple destinations by one <Send> as well. In this case destinations can be enumerated inside the body as “FixSession”, “Client”:

```
<Rule>
  <Source>
    <Client Name=".*"/>
  </Source>
  <Action>
    <Send>
      <FixSession SenderCompID="Sender1" TargetCompID="Target1"/>
      <FixSession SenderCompID="Sender2" TargetCompID="Target2"/>
      <Client Name="TestClient1"/>
      <Client Name="TestClient2"/>
    </Send>
  </Action>
</Rule>
```

Below you can see the example of routing to multiple destinations identified by source identifiers (e.g. “Name” attribute of <Send> tag):

```
<Rule>
  <Source Name=".*"/>
        <Action>
                <Send Name="Session1"/>
                <Send Name="ClientIdentifer">
                        <Client Name="TestClient1"/>
                        <FixSession SenderCompID="Sender2" TargetCompID="Target2"/>
                </Send>
        <Action>
</Rule>
```

Note: the example also demonstrates the embedded destinations usage in the <Send> rule for source identifier “ClientIdentifier”; rules of such kind are allowed in the routing rules.

Transformation rules

You can use SetField, Convert, MoveField, RemoveField or Script tags to change a message.
Example:

```
<Rule>
  <Source>
    <Client Name="TestClient"/>
```

```
  </Source>
  <Condition>
    <MatchField Field="35" Value=".*"/>
    <MatchField Field="571" Value="IST-\\d+"/>
  </Condition>
    <Action>
      <MoveField SourceField="32" TargetField="48" />
      <Script Language='Javascript' Field="" LengthField="" FileName="sc_2_3.js" >
          <Param Name="Acct">JBDFLYX</Param>
      </Script>
      <Convert TargetProtocol="FIX.4.3" SourceProtocol="">
           <SetField Field="35" Value="abc" />
           <RemoveField Field="50" />
      </Convert>
    </Action>
</Rule>
```

If more than one transformation is specified the next transformation is applied to the result of the previous transformation.

Mixed rules

There are no restrictions for the number of Action elements (acts) of different types. In this case they are executed in the order of appearance.
Example:

```
<Rule>
  <Source>
    <Client Name="TestClient"/>
  </Source>
  <Condition>
    <MatchField Field="35" Value="C"/>
  </Condition>
  <Action>
    <SetField Field="57" Value="TargetSubID"/>
    <Send>
      <FixSession SenderCompID="TestSender" TargetCompID="TestTarget"/>
    </Send>
    <SetField Field="57" Value="AdminTargetSubID"/>
    <Send>
      <FixSession SenderCompID="TestSender" TargetCompID="AdminTestTarget"/>
    </Send>
  </Action>
</Rule>
```

This will result in the scenario below:

1) *Message is duplicated: SrcMsg → DupMsg*
2) *For DupMsg: Field 57 is set to "TargetSubID"*

3) *DupMsg is set to the defined FIX Session ID*
4) *Message is duplicated: DupMsg → DupDupMsg*
5) *For DupDupMsg: Field 57 is set to "AdminTargetSubID"*
6) *DupDupMsg is set to the defined FIX Session ID*

Note: The message will be duplicated according to the number of Send actions in the rule.

Syntax

Rule structure

<Rule> element has a single “Enabled” attribute. <Rule Enabled=”false”> disables rule parsing by Business Layer. By default the attribute value has a true value.
Each rule consists of:

- *Source* (required section)
- *Condition* (implied section)
- *Action* (required section)

Note that XML comments cannot be inside <Rule> structure, only outside the rule. If the formed rules do not meet the syntax requirements, FIXEdge stops loading and records an error message in the log file.

Source section

Source can be a FIX Session, which is a pair of SenderCompID and TargetCompID or a Client, which is denoted by client Name. *Source* section may contain the following elements:

- *Client* – defines Transport Client as source:

```
<Source>
        <Client Name="TestClient"/>
</Source>
```

- *FIXSession* – defines FIX session as source; FIX session is identified by SenderCompID and TargetCompID pair:

```
<Source>
      <FixSession SenderCompID="Sender" TargetCompID="Target"/>
</Source>
```

- *Name* attribute – defines the identifier of FIX session or Transport Client as source (Please refer to Source identification for details):

```
<Source Name="SourceID"/>
```

Condition section

Condition is a set of pre-defined criteria. The application checks FIX messages on the criteria and if *Condition* is met *Action* is to be executed; otherwise *Action* is ignored.
Condition section may contain the following criteria:

- *EqualField* – compares two values:

```
<EqualField Field="35" Value="D" />
```

- *NotEqualField* – compares two values and returns true when they are not equal:

```
<NotEqualField Field="35" Value="asd"/>
```

- *MatchField* – compares two values treating value as a regular expression:

```
<MatchField Field="35" Value=".*"/>
```

- *NotMatchField* – compares two values as a regular expression and returns true when they are not equal:

```
<MatchField Field="35" Value="[1-9]$"/>
```

- *ExistField* – checks the field presence in message:

```
<ExistField Field="35"/>
```

- *NotExistField* – checks the field presence in message, returns true if it does not exist:

```
<ExistField Field="35"/>
```

- *Script* – applies JavaScript condition to message:

```
<Script Language="JavaScript" FileName ="testScript.js"/>
```

- *MatchMessage* – applies regular expression to message string representation:

```
<MatchMessage Value=".*?55=10.*?"/>
```

- *NotMatchMessage* – applies regular expression to message string representation and returns true when regular expression is not matched:

```
<NotMatchMessage Value=".*?55=10.*?"/>
```

- *EqualDestination* – locates message destination session according to the specified automatic routing strategy and returns true when the located destination session is equal to the one specified in action

```
<EqualDestination Strategy="OrderFlowStrategy100" Value="Client"/>
```

- *NotEqualDestination* - locates message destination session according to the specified automatic routing strategy and returns true when the located destination session is not equal to the one specified in action

```
<NotEqualDestination Strategy="OrderFlowStrategy100" Value="Client"/>
```

- *MatchDestination* - locates message destination session according to the specified automatic routing strategy and returns true when the located destination session is matched to the specified pattern

```
<MatchDestination Strategy="OrderFlowStrategy100" Value="FIXSession.*"/>
```

- *NotMatchDestination* - locates message destination session according to the specified automatic routing strategy and returns true when the located destination session is not matched to the specified pattern

```
<NotMatchDestination Strategy="OrderFlowStrategy100" Value="FIXSession.*"/>
```

It is recommended to use *EqualField* (*NotEqualField*) rather than *MatchField* (*NotMatchField*), when applicable. Although *MatchField* is more flexible, it may decrease the performance.
Condition section is not required.

Action section

Action denotes the instructions that must be performed when a Rule is being applied to a message. *Action* is always a required section of any Rule. It must contain at least one instruction. *Action* section may contain the following instruction types:

- *Send* – sends a message to a specified destination:

```
<Send><FixSession SenderCompID="Sender" TargetCompID="Target"/></Send>
```

- *SetField* – assigns the given value to a specified field:

```
<SetField Field="50" Value="Ordinary Sender SubID"/>
```

- *CopyField* – copies the value from SourceField to TargetField:

```
<CopyField SourceField="55" TargetField="58" IsRequiredField="Y"/>
```

- *MoveField* –moves the value from SourceField to TargetField:

```
<MoveField SourceField="50" TargetField="53"/>
```

- FormatSymbol - splits or assembles the value of Symbol tag

```
<FromatSymbol ActionType="Assemble" TrgSeparator="." />
```

- *RemoveField* – removes a specified field from message:

```
<RemoveField Field="50" />
```

- *Script* – executes instructions from Javascript or XSLT file with given parameters:

```
<Script Language="XSLT" Field="213" LengthField="212" FileName="file.xsl"><Param
Name="Acct">JBDFLYX</Param></Script>
```

- *Convert* – converts a message to a specified FIX protocol dialect:

```
<Convert TargetProtocol="FIX.4.3" SourceProtocol=""/>
```

- *Transform* – makes a new message of a specified message type based on a source message:

```
<Transform TargetMsgType="9"/>
```

- *HandlerAction* – handles a message by the defined handler:

```
<HandlerAction Name="MsgProcessor" />
```

- *CreateReject* – creates a corresponding reject message for the routed message:

```
<CreateReject RejectType="session"/>
```

- *ModifyField* – applies a specified plug-in method to the routed message:

```
<ModifyField Field="11" Rule="Brass.ClOrdId.Converter.Brass2Source()" />
```

- *FormatSymbol* – modifies the value of Symbol and SymbolSfx tags. The action is able to merge Symbol and SymbolSfx tags into the value of Symbol tag or split the value of Symbol tag into Symbol and SymbolSfx tags:

```
<FormatSymbol ActionType="Assemble" TrgSeparator="_" />
```

- *SaveToHistory* – stores data from the handled message to history storage:

```
<SaveToHistory Name="Orders"/>
```

- *ClearHistory* – starts history storage clearing, obsolete records are erased:

```
< ClearHistory Name="Orders"/>
```

- *CreateReject* – creates a corresponding reject message for the handled message:

```
<CreateReject RejectType="session"/>
```

- *ProcessNetStatusRequest* – creates a corresponding NetStatusRespone FIX message for the handled NetStatusRequest message:

```
<ProcessNetStatusRequest/>
```

- *CreateNotification* – creates a notification Email FIX message with specified parameters:

```
<CreateNotification Category="N" Reason="Test Reason"/>
```

- *SystemCommand* – executes system command or script:

```
<SystemCommand Command="runSystemCommand.bat"/>
```

- *ConvertToBusinessReject* – converts SessionLevelReject FIX message into the corresponding BusinessReject FIX message:

```
<ConvertToBusinessReject/>
```

- *StrategySend* – applies a specified automatic routing strategy to the handled message, locates a destination session and sends the message to the required session:

```
```

```
<StrategySend Name="OrderFlowStrategy100"/>
```

Source identification

Identifier represents the name of a source in a routing rule specified for some FIX session. If the source identifier is already registered in server configuration and related sessions are established, FIXEdge will define the way of message routing between FIX sessions and TA clients. The source identifier will be unbound as FIX session name if explicit unlink is executed (using XML command for example).
The routing table determines the path for incoming FIX message to be routed:

Source (source name)	Condition	Action (destination name)
Source name - [Session Name] or [ClientID] Destination name - [ClientID] or [Session Name]	35 ::= AE 571 ::= IST-\\d+	Send to [ClientID]

here **source name** and **destination name** define the direction of message routing.

According to this routing table the rule will look in the following way:

```
<Rule>
        <Source Name = “SourceName”/>
        <Condition>
                <MatchField Field="35" Value=".*"/>
                <MatchField Field="571" Value="IST-\\d+"/>
        </Condition>
        <Action>
                <Send Name = “ClientID”/>
        </Action>
</Rule>
```

The routing table applies this rule to all incoming messages from session with “SessionName” identified with “SourceName” and send them to source of TA client “ClientID”.

Source identifier has the following restrictions:

- The name of source identifier must be unique for each FIX session;
- The source identifier of TA client must be represented as Client ID;
- It must contain characters with codes from ASCII [30] to ASCII [128];
- It must begin with a letter, digit or underlining symbol ‘_’ ;

Whitespace characters in the beginning or in the end of name will be ignored by rule parsing.

FIXEdge events

Session events

FIXEdge router provides the user with a possibility to establish a set of Business Rules that handles FIXEdge events:

- New session was created.
- Session was destroyed.
- Unable to route message to a specified target.
- Session level reject was received in response to the sent application message.
- Routing rule failed during execution.
- Source section may have <FixSession SenderCompID="" TargetCompID=""/> element to identify a session that launches an event.
- Source section may have <Client Name=""/> element to identify a client that launches an event.
- Source section of any event may have <Source Name=""/> attribute of source identifier that launches an event (see Source identification for details).

Create session event

FIXEdge launches a create session event in case a new session is created. The XML element *CreateSessionEvent* rule consists of the following sections:

- *Source*
- *Condition*
- *CreateSessionAction*

Source section of *CreateSessionEvent* is the same as *Source* of the Rule XML element.
Source is a required section.
Condition section of *CreateSessionEvent* is the same as *Condition* of the Rule XML element.
Condition is not a required section.
CreateSessionAction section denotes the actions that must be applied to message.
CreateSessionAction is a required section and it must contain at least one instruction.
CreateSessionAction section may contain the following instructions:

- *Script* – applies JavaScript condition to message:

```
<Script Language="JavaScript" FileName ="testScript.js"/>
```

- *AcceptSession* – accepts session creation. *CreateSessionAction* section may contain this action only once:

```
<AcceptSession />
```

- *RejectSession* – declines session creation. *CreateSessionAction* section may contain

this action only once:

```
<RejectSession />
```

Destroy session event

FIXEdge launches a destroy session event in case the session is closed. The XML element *DestroySessionEvent* rule consists of the following sections:

- *Source*
- *Condition*
- *DestroySessionAction*

Source section of *DestroySessionEvent* is the same as *Source* of the Rule XML element. *Source* is a required section.
Condition section of *DestroySessionEvent* is the same as *Condition* of the Rule XML element. *Condition* is not a required section.
DestroySessionAction section denotes the actions that must be applied to a message. *DestroySessionAction* is a required section and it must contain at least one instruction. *DestroySessionAction* section may contain the following instructions:

- *Script* – applies JavaScript condition to message:

```
<Script Language="JavaScript" FileName ="testScript.js"/>
```

Unable to route message event

FIXEdge launches an unable to route message event in case message routing failed and FIXEdge is unable to route message to a specified target session or client. The XML element OnUndeliveredMessageEvent rule consists of the sections:

- *Source*
- *Condition*
- *Action*

Source section of *OnUndeliveredMessageEvent* is the same as *Source* of the Rule XML element. *Source* is a required section.
Condition section of *OnUndeliveredMessageEvent* is the same as *Condition* of the Rule XML element. *Condition* is not a required section.
Action section of *OnUndeliveredMessageEvent* is the same as *Action* of the Rule XML element. *Action* is a required section.

Session level reject is received in response to the sent application message

FIXEdge launches a session level reject event in case SessionLevelReject FIX message is received. The XML element *OnSessionLevelRejectEvent* rule consists of the following sections:

- *Source*
- *Condition*
- *Action*

Source section of *OnSessionLevelRejectEvent* is the same as *Source* of the Rule XML element. *Source* is a required section.
Condition section of *OnSessionLevelRejectEvent* is the same as *Condition* of the Rule XML element. *Condition* is not a required section.
Action section of *OnSessionLevelRejectEvent* is the same as *Action* of the Rule XML element. *Action* is a required section.

Routing rule failed during execution
FIXEdge launches a routing rule failed event in case of BL Rule failure during message handling, when FIXEdge is unable to route message according to BL Rule. The XML element *OnRuleFailEvent* rule consists of the following sections:

- *Source*
- *Condition*
- *Action*

Source section of *OnRuleFailEvent* is the same as *Source* of the Rule XML element. *Source* is a required section.
Condition section of *OnRuleFailEvent* is the same as *Condition* of the Rule XML element. *Condition* is not a required section.
Action section of *OnRuleFailEvent* is the same as *Action* of the Rule XML element. *Action* is a required section.

Plug-ins

FIXEdge offers a flexible solution for modification of FIX message partial fields - *Plugins*. *Plugins* provide an open interface and can be written in any programming language that allows creating modules in Microsoft Windows DLL format. The custom Plugins mechanism can be used in case Business Layer's built-in mechanisms and engines cannot perform a specified task. For example when conversion data must be stored from call to call or when its data must be persistent.

Plugins section may contain one or more Plugin definitions. Each Plugin definition contains a set of plugin parameters. Plugin element has a required '*Plugin Path*' attribute that contains both path and name of plugin DLL. Plugin parameters are defined in *Param* element with a required '*Param Name*' attribute.
ModifyField action refers to the defined plugin and executes plugin method for the routed message.

Handlers

FIXEdge offers a flexible solution for FIX message processing - *Handlers*. *Handlers* provide an open interface and can be written in any programming language that allows creating modules in Microsoft Windows DLL format. The custom Handlers mechanism can be used in case Business Layer's built-in mechanisms and engines cannot perform a specified task. For example when data must be stored from message to message or when data must be persistent. FIXEdge loads Handlers only once: when it parses the XML file.

DllHandlers section may contain one or more Handler definitions. Each Handler definition has the following attributes:

- *Handler Name* – unique handler name that will be used in *HandlerAction*. The attribute is required.
- *Description* – handler description. The attribute is required.
- *DllName* – contains the path and file name of the Handler's DLL. The attribute is required.
- *VerifyHandlersVersion* – specifies whether FIXEdge must verify the version of Handler interface when loading this handler. It prevents the "dll hell" problem. The attribute is required.

HandlerAction action refers to the defined Handler and initiates message handling by a specified Handler.

Histories

FIXEdge offers a flexible solution for storing information that goes through the Business Layer of FIXEdge – Histories. Business Layer supports the following types of Histories:

- ODBCHistory – allows storing information in the database
- FileHistory - allows storing information in the persistent file-based storage
- MemoryHistory - allows storing information in the transient memory-based storage

Histories can be used in routing rules only when they are defined in the BL rules XML file. Each history supports the following attributes:

- Name – name of history, used to refer to history from XML actions or JavaScript functions. The attribute is required.
- StorageType – type of history, the valid values are: “ODBC”, “File”, “Memory”. The attribute is required.
- ClearTime –defines when every-day clear procedure is started. The format is HH:MM:SS. Clear procedure erases obsolete records.
- Additional attributes depend on the history type.

The Business Layer of FIXEdge can be extended with the set of actions to work with defined Histories from XML rules and JavaScript. The following XML actions must be defined:

- SaveToHistory – saves the data retrieved from FIX message to history.
- ClearHistory – erases obsolete records from history.

Following JavaScript functions defined:

- saveToHistory – saves data to history
- getFromHistory – locates a record in history by key and retrieves the value of record field
- getRecordFromHistory – locates a record in history by key and returns it

ODBC History

ODBCHistory supports the following additional attributes:

- TableName –name of table in database. History will be reflected in this table. The attribute is required
- ColumnSize –default size of history string fields.
- ConnectionString –ODBC connection string. The attribute is required

ODBCHistory supports the following types of fields:

- Int
- Float
- Date
- DateTime
- String – default field type

Example of ODBCHistory definition:

```
//Defines history “SampleODBCHistory” that reflects on the Orders table in database SampleBase.
// Following FIX message tags mapped into the Orders table fields:
```

```
//    ClOrdID(11) -> ClOrdID, type – string
//    SenderCompId(49) -> SenderCompId, type – string[512]
//    TargetCompId(56) -> TargetCompId, type – string
//    SettlDate(64) -> SettlDate, type – date
//  History maps the record's ExpireDateTime attribute to the database field TransactTime
<History Name="SampleODBCHistory"
              StorageType="ODBC"
              TableName="SampleBase.dbo.Orders"
              ClearTime="22:00:00"
              ColumnSize="256"
              ConnectionString="DSN=SampleBase;UID=test;PWD=test123;">
            <KeyField ColumnName="ClOrdID">11<KeyField>
            <Field ColumnName="SenderCompID" ColumnSize="512">49</Field>
            <Field ColumnName="TargetCompID">56</Field>
      <Field ColumnName="SettlDate" DataType="Date" ColumnSize="10">64</Field>     <Field
ColumnName="TransactTime">ExpireDateTime</Field>
</History>
```

File History

FileHistory supports the following additional attributes:

- WorkingDirectory – path to stored history files. The attribute is required.
- StorageFileName – name of history storage file. The attribute is required.

Example of FileHistory definition:

```
//Defines history "SampleFileHistory", history saves data to the file .\logs\SmplFileHistory
// The ClOrdID(11), SenderCompId(49) and TargetCompId(56) FIX tags used as a record composite key
<History Name="SampleFileHistory"
        WorkingDirectory=".\logs"
              StorageType="File"
              StorageFileName="SmplFileHistory"
              ClearTime="22:00:00">
            <KeyFields>11, 49, 56<KeyFields>
            <Fields> 99,18,63,64,528,529,100</Fields>
            <Field>ExpireDateTime</Field>
</History>
```

Memory History

MemoryHistory does not require additional attributes.

Example of MemoryHistory definition:

```
//Defines history "SampleMemHistory"
// The ClOrdID(11), SenderCompId(49) and TargetCompId(56) FIX tags used as a record composite key
<History Name="SampleMemHistory"
              StorageType="Memory"
```

```
            ClearTime="22:00:00">
          <KeyFields>11, 49, 56<KeyFields>
          <Fields> 99,18,63,64,528,529,100</Fields>
          <Field>ExpireDateTime</Field>
</History>
```

Examples

1. Messages from all FIX sessions are sent to one client with ‘MQClient’ ID. Messages from the session with ‘MQClient’ ID are sent to FIX session according to SenderCompID and TargetCompID set in message.

```
<?xml version="1.0" encoding="UTF-8"?>
<FIXEdge>
   <BusinessLayer>
      <Rule Description="Routing rule for the incoming messages">
         <Source>
            <FixSession SenderCompID=".*" TargetCompID=".*"/>
         </Source>
         <Action>
            <Send>
               <Client Name="MQClient"/>
            </Send>
         </Action>
      </Rule>
      <Rule Description=”Routing rule for the outgoing messages”>
         <Source>
            <Client Name="MQClient"/>
         </Source>
         <Action>
            <Send>
               <FixSession />
            </Send>
         </Action>
      </Rule>
   </BusinessLayer>
</FIXEdge>
```

2. Messages are routed to different sessions depending on ExDestination. If ExDestination=”ISE” messages are sent to the client with “ISE” ID. If ExDestination=”AMEX” messages are sent to the client with “AMEX” ID, otherwise messages are sent to the client with “Unknown” ID. All messages from all clients are sent back to the FIX session.

```
<?xml version="1.0" encoding="UTF-8"?>
<FIXEdge>
   <BusinessLayer>
      <Rule Description="Routing rule for the AMEX messages">
         <Source>
```

```
            <FixSession SenderCompID="Foo" TargetCompID="Bar"/>
         </Source>
         <Condition>
            <MatchField Field="100" Value="AMEX"/>
         </Condition>
         <Action>
            <Send>
               <Client Name="AMEX"/>
            </Send>
         </Action>
      </Rule>
      <Rule Description="Routing rule for the ISE messages">
         <Source>
            <FixSession SenderCompID="Foo" TargetCompID="Bar"/>
         </Source>
         <Condition>
            <MatchField Field="100" Value="ISE"/>
         </Condition>
         <Action>
            <Send>
               <Client Name="ISE"/>
            </Send>
         </Action>
      </Rule>
      <Rule Description="Routing rule for other messages">
         <Source>
            <FixSession SenderCompID="Foo" TargetCompID="Bar"/>
         </Source>
         <Condition>
            <NotMatchField Field="100" Value="AMEX|ISE"/>
         </Condition>
         <Action>
            <Send>
               <Client Name="Unknown"/>
            </Send>
         </Action>
      </Rule>
      <Rule Description=”Routing rule for the outgoing messages”>
         <Source>
            <Client Name=".*"/>
         </Source>
         <Action>
            <Send>
               <FixSession SenderCompID="Foo" TargetCompID="Bar"/>
            </Send>
         </Action>
      </Rule>
   </BusinessLayer>
</FIXEdge>
```

3. There are no restrictions for the number of Action elements (acts) of different types. In this case they are executed in the order of appearance.

```
<Rule>
  <Source>
    <Client Name="TestClient"/>
  </Source>
  <Condition>
    <MatchField Field="35" Value="C"/>
  </Condition>
  <Action>
    <SetField Field="57" Value="TargetSubID"/>
    <Send>
      <FixSession SenderCompID="TestSender" TargetCompID="TestTarget"/>
    </Send>
    <SetField Field="57" Value="AdminTargetSubID"/>
    <Send>
      <FixSession SenderCompID="TestSender" TargetCompID="AdminTestTarget"/>
    </Send>
  </Action>
</Rule>
```

This will result in the scenario below:

1. Message is duplicated: SrcMsg → DupMsg
2. For DupMsg: Field 57 is set to "TargetSubID"
3. DupMsg is sent to the defined FIX Session ID
4. Message is duplicated: DupMsg → DupDupMsg
5. For DupDupMsg: Field 57 is set to "AdminTargetSubID"
6. DupDupMsg is sent to the defined FIX Session ID

Note: The message will be duplicated according to the number of Send actions in the rule.

4. Reject creation of the session with “Test” client name

```
<CreateSessionEvent>
  <Source>
    <Client Name="Test"/>
  </Source>
  <CreateSessionAction>
    <RejectSession />
  </CreateSessionAction>
</ CreateSessionEvent>
```

5. Handle Unable to route message event – route all non-routed messages to the client with “Recycle” ID.

```
< OnUndeliveredMessageEvent >
  <Source>
```

```
    <Client Name=".*"/>
  </Source>
  < Action >
      <Send>
        <Client Name=" Recycle "/>
      </Send>
  </ Action >
</ OnUndeliveredMessageEvent >
```

6. Define plugin and use it to modify the value of tag 11

```
 <Plugins>
  <Plugin Path=".\FIXServer\bin\PluginConverter.dll">
      <Param Name="MinBranchCode">AAAA</Param>
      <Param Name="MaxBranchCode">ZZZZ</Param>
      <Param Name="WorkingDirectory">.\FIXServer\bin\plugin\</Param>
      <Param Name="StorageName">PluginConverter</Param>
  </Plugin>
 </Plugins>
 <Rule Description="Routing rule for the incoming messages">
      <Source>
          <FixSession SenderCompID="Client" TargetCompID="FixEdge"/>
      </Source>
      <Action>
          <ModifyField Field="11" Rule=" PluginConverter.Source2Plugin()" />
          <Send>
              <FixSession SenderCompID="FixEdge" TargetCompID="Client2"/>
          </Send>
      </Action>
 </Rule>
```

7. Define handler and use it to process messages from FIX session

```
 <DllHandlers>
      <Handler Name="TestHandler"
          Description="Test Handler description"
          DllName="./TestHandler.dll"
           VerifyHandlersVersion="true" />
 </DllHandlers>
  <Rule Description="Routing rule for the incoming messages">
      <Source>
          <FixSession SenderCompID="Client" TargetCompID="FixEdge"/>
      </Source>
      <Action>
          < HandlerAction Name="TestHandler" />
          <Send>
              <FixSession SenderCompID="FixEdge" TargetCompID="Client2"/>
          </Send>
      </Action>
 </Rule>
```

Scripts

FIXEdge offers a flexible solution for routing and transferring of FIX messages that go through the Business Layer of FIXEdge and data manipulations. In addition to XML Rules, there are two ways to enhance the flexibility of message transferring and data manipulation. Those ways are scripting with Javascript and XSLT. The proper usage of this feature requires scripts written in one of the scripting languages and assigned to the Script instruction of Action section in a Business Rule.
The scripting subsystem gives the user the following advantages:

- Saves the time spent on business logic implementation
- Flexibility and simplicity of modification
- Does not cause significant losses in performance

You can find more information about Javascript and XSTL scripting can in

FixEdge_BR_Scripting_UserGuide.html.

Business Rule scripting with Javascript

It is enough to mention that the script is a “Javascript” in the Script tag, in order to run the script and perform the proper transformations.

FIXEdge Javascript act provides the user with all the standard Javascript features and functionality. In addition to this, it enables the user to use functions that provide access to FIX message inside data, allows manipulations with fields and groups as well as whole FIX messages.
Predefined FIX date formats:

- HHMMSSUtc for the UTC "HH:MM:SS" date format
- HHMMSSsssUtc for the UTC "HH:MM:SS.sss" date format
- YYYYMM for the "YYYYMM" date format
- YYYYMMDD for the"YYYYMMDD" date format
- YYYYMMWW for the"YYYYMMWW" date format, where WW - number of week "w1"..."w4"
- DATETIMEUtc for the UTC "YYYYMMDD-HH:MM:SS" date format
- DATETIMEsssUtc for the UTC "YYYYMMDD-HH:MM:SS.sss" date format
- YYYYMMDDLmd for the local market "YYYYMMDD" date format

A trivial example of a complete JS script for a Business Rule is given below:

```
//take the value the 11th tag from the fix message and get the last 4
//characters from it
//assign the new value for the tag 11 with format BBBBSSSSCCYYMMDD
//where
//BBBB - any 4 characters
//SSSS - last 4 characters from the old value of field 11
//Ð¡Ð¡ - century
```

```
//YY- year
//MM - month
//DD - day
field_11 = getStringField(11);
if(field_11.length() < 4)
  return;
last4 = field_11.slice(field_11.length() - 4);
currDate = getCurrentDateStr(YYYYMMDDUtc);
century = currDate.slice(1, 3);
century++;
result = "bbbb" + last4 + century + currDate.substr(3);
setStringField(11, result);
```

Error handling during script execution

In case of syntax error in script its execution will be terminated and failed. There is no way to handle syntax errors in script during script execution. The user is able to stop script execution using throw statement, the result in this case will be the same. The user must be able to verify and handle logical errors using verification functions (like isNaN, isMessageValid(), etc) in condition statements.

Business Rule scripting with XSLT

When a more complicated operation than a standard Business Rule can offer is required for field transformation, the XSLT script action can be used. The user should specify a certain FIX Field [and Field Length] in the Script instruction. The rules processor will extract the field value, apply the specified transformation and assign the obtained result to the field.

XSLT Script instruction in the Action section of a Business Rule:

```
<Script Language="XSLT" Field="213" LengthField="212" FileName="script1.xslt" >
   <Param Name="AcctParam">'JBDFLLC'</Param>
</Script>
```

This instruction causes the rule processor to get the value of FIX field 213, loads XSLT script from *script1.xslt* file, applies the transformation to the value passing *AcctParam* parameter there and assigns the resulting value to field 213.

LengthField attribute is optional. If it is specified the rule processor sets the length to the result. This attribute must be specified if the field has FIX type Raw Data.

The specification for XSLT can be found at W3C . Apache Xalan is used as internal XSLT processor.

Example:

Messages from all FIX sessions are sent to one client with ‘MQClient’ ID. Messages from the session with ‘MQClient’ ID are sent to FIX session according to SenderCompID and

TargetCompID set in message.

```
<?xml version="1.0" encoding="UTF-8"?>
<FixEdge>
   <BusinessLayer>
      <Rule Description="Routing rule for the incoming messages">
         <Source>
            <FixSession SenderCompID=".*" TargetCompID=".*"/>
         </Source>
         <Action>
            <Send>
               <Client Name="MQClient"/>
            </Send>
         </Action>
      </Rule>
      <Rule Description=”Routing rule for the outgoing messages”>
         <Source>
            <Client Name="MQClient"/>
         </Source>
         <Action>
            <Send>
               <FixSession />
            </Send>
         </Action>
      </Rule>
   </BusinessLayer>
</FixEdge>
```

Default routing strategies

There are two forms of default routing:

< DeliverToStrategy> element specifies the routing destination that has been specified beforehand inside the source FIX message.
The following properties of the element are required:
Property **Name** – contains the name of element to identify in factory rules.
Property **TargetField** – contains a tag number with routing destination. Actually, there may be tag 128 <DeliverToCompID>.
SourceField – contains a tag number with FIX message source and provides the possible back way relation. The property is not required.

```
<DeliverToStrategy Name="DeliverToStrategy115128" SourceField="115" TargetField="128"/>
```

Note: Business Layer takes the value from tag 49 <Sender> in case **SourceField** tag is not found in FIX message or is not specified in a rule. The direction of message routing is defined by session name that is a pair of **Source|Target** (or **Sender|Target** when **SourceField** is not found). The session with a specified pair of **Source|Target** should be registered in FIXEdge, otherwise a message will be rejected.

<OrderFlowStrategy> element defines the orders flow to destination that can be found in the history of orders.
The following properties of element are required:
Property **Name** – contains the name of element to identify in factory rules.
Property **History** – contains the name of History element, where the order flow strategy will refer to specify the routing destination.
Property **DestinationField** – contains a tag number with destination. For example, tag 100 <ExDestination>.

```
<OrderFlowStrategy Name="OrderFlowStrategy100" History="Orders" DestinationField="100" />
```

Note: **OrderFlow** strategy defines the destination of FIX message routing by source identifier only. Session with source identifier that in the destination field should be registered in FIXEdge, otherwise an order will be rejected.

History used in the OrderFlow strategy must contain the following required fields:
key fields: ClOrdID(11), CrossID(548), ListID(66)
common fields: SenderCompID(49), TargetCompID (56), OrdStatus (39), 'DestinationField' number in strategy description (default 100)

Factory_OrderHistory.xml

```
<History Name="Orders"
      StorageType="File"
      CacheRecords="5"
      ClearTime="23:50:00"
      WorkingDirectory="L:/Program Files/B2Bits/FIXEdge/FixEdge1/log/"
      StorageFileName="BLHistory_Orders.log">
      <KeyFields>11,548,66</KeyFields>
      <Fields>49,56,100,1,39,439,440,167</Fields>
</History>
```

Examples

Factory_Strategy.xml

```
<OrderFlowStrategy Name=”OrderFlowStrategy100” History=”Orders” DestinationField=”100” />

<DeliverToStrategy Name=”DeliverToStrategy115128” SourceField=”115” TargetField=”128”/>
```

Factory_OrderFlowRule.xml

```
<Rule Enabled=”true” Description=”Factory rule for new order”>
      <Source>
            <Session Name=”.*” />
      </Source>
      <Condition>
            <MatchField Tag=”35” Value=”D|AB|G|AC|8|9” />
      </Condition>
      <Action>
            <StrategySend Name=”OrderFlowStrategy100” />
      </Action>
</Rule>
```

CME_OrderRue.xml

```
<Rule Enabled=”false” Description=”Override for new order factory rule, when send to CME”>
      <Source>
            <Session Name=”.*” />
      </Source>
      <Condition>
            <MatchField Tag=”35” Value=”D|AB” />
            <EqualField Tag=”100” Value=”CME” />
```

```
    </Condition>
    <Action>
        <Convert TargetProtocol=”CME_FIX42” />
        <SaveToHistory Name=”Orders”>
        <Send>
            <Session Name=”CME”>
        </Send>
    </Action>
</Rule>
```

CME_ReplaceRule.xml

```
<Rule Enabled=”false” Description=”Override for replace factory rule, when send to CME”>
    <Source>
        <Session Name=”.*” />
    </Source>
    <Condition>
        <MatchField Tag=”35” Value=”G|AC” />
                                                <MatchDestination Strategy=”OrderFlowStrategy100”
                                    Value=”CME” />
    </Condition>
    <Action>
        <Convert TargetProtocol=”CME_FIX42” />
        <StrategySend Name=”OrderFlowStrategy100” />
    </Action>
</Rule>
```

Factory_DeliverToMessageRule.xml

```
<Rule Enabled=”true” Description=”Factory rule for strategy based on DeliverTo”>
    <Source>
        <Session Name=”.*” />
    </Source>
    <Condition>
        <MatchField Tag=”35” Value=”c|d” />
    </Condition>
    <Action>
        <StrategySend Name=”DeliverToStrategy115128” />
    </Action>
</Rule>
```

Contact us

sales@btobits.com
Phone: +1-888-378-0666

Global Headquarters
US Client Support and Delivery Center
EPAM Systems, Inc
41 University Drive
Suite 202
Newtown, PA 18940
Phone: +1-267-759-9000
Fax: +1-267-759-8989