Business Rules Guide

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. This file is loaded into the memory during FIXEdge starting. The installation package contains the business layer configuration file with rules that allow to test the FIXEdge functionality, but it needs to be configured for client's business needs in accordance with the recommendations described in this section.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 FIXICC or by editing the file directly. They can also be reloaded immediately from FIXICC.

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

All described above elements belong to the element <BusinessLayer>.

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

Rules processing order

Due to the specifics of the Business Layer Configuration File more than one rule can be applied to message. It is also possible to route one message to multiple destinations or to the same destination multiple times.

There are several items to mention here:

- All rules are applied independently.
- Rules are not necessarily performed at the order specified in BL Configuration File. The order is not predefined.
- If a **set** of transformation rules is applied to the same message, each rule receives its own copy of the original message and is applied to it.
- Actions defined in the rule are applied in the specified order, except the cases listed below:
    - If the rule contains sending instructions, they are triggered and prepared at the specified in BL xml file place but are performed (*meaning sent*) after all other actions from the <action> element, except DisconnectSession or TerminateSession session instructions. See Routing Rules and Session Events: Mixed rules for an example.
    - If the rule contains a **set** of sending instructions, they are not necessarily performed (*meaning sent*) at the order specified in BL Configuration File. The order is not predefined.
    - If the rule contains DisconnectSession or TerminateSession session instructions, they are performed at the very end, after all sending instructions.

Rule

<Rule> element has couple of attributes: “Enabled” and "Description". <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)

Refer to Rule element article for detailed description.

Default Rule

<DefaultRule> element defines special type of rules, which contains section <Action> only. Executed if no other rule is applied. Located at the end of Business Layer Configuration file. Usually contains action <DoNothing/>.

```
<FIXEdge>
  <BusinessLayer>
     <!-- ... -->
     <DefaultRule>
       <Action>
         <DoNothing/>
       </Action>
     </DefaultRule>
  </BusinessLayer>
</FIXEdge>
```

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.

Refer to Session Events for detailed description.

Plug-ins

```
<FIXEdge>
 <BusinessLayer>
   <Plugins>
     <Plugin Path=".
/Brass.dll">
       <Param Name="
Param1"/>
     </Plugin>
   </Plugins>
 </BusinessLayer>
</FIXEdge>
```

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 is optional and may contain one or more <plugin> definitions.

Plugin element

Each <Plugin> element has:

Attribute '*Path*'	Mandatory. Defines path and name of plugin DLL
Set of plugin parameters	Optional. Plugin parameters are defined via <*Param*> *xml*-element with a required '*Name*' attribute.

Usage: Rule's instruction *ModifyField* refers to the defined plugin and executes plugin method for the routed message.

Param element

Param element is used in the configuration file within <Plugin> and <Script> elements to define their parameters.

Attributes:

Name	Mandatory. Defines name of parameter
Value	Optional. Defines parameter value by default

Handlers

```
<FIXEdge>
 <BusinessLayer>
   <DllHandlers>
      <Handler
VerifyHandlersVersion="true"
               DllName=".
/PSHandlerPlugin.dll"
               Description="Test
Handler description"
               Name="TestHandler"
/>
   </DllHandlers>
 </BusinessLayer>
</FixEdge>
```

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 business layer configuration file.

Optional <DllHandlers> section may contain one or more <handler> definitions.

Handler element

Mandatory attributes:

Name	Unique handler name that will be used in *HandlerAction*
Description	Handler description
DllName	Contains the path and file name of the Handler's DLL
VerifyHandlersVersion	Specifies whether FIXEdge must verify the version of Handler interface when loading this handler. It prevents the "dll hell" problem Valid values: "true"|"false"

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

See part FIXEdge API for BL Handlers for details.

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> element has the common attributes:

Name	Mandatory. Name of history, used to refer to history from XML actions or JavaScript functions.
StorageType	Mandatory. Defines type of history. Valid values: "ODBC" | "File" | "Memory".
ClearTime	Optional. Defines when every-day clear procedure is started. The format is HH:MM:SS. Clear procedure erases obsolete records.

and the additional attributes in 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 are available:

- 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
- updateHistory – modifyes fields in the 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
- removeRecordFromHistory - locates a record in history by key and removes it
- removeRecordFromHistoryByCompositeKey - locates a record in history by composite key and removes it

ODBC History

Each ODBC Histoty defines regulations of storage FIX message tags within the particular table of database. ODBCHistory supports the following additional attributes:

- TableName – name of table in database. History will be reflected in this table. The attribute is required.
- ColumnSize – Optional. Default size of history string fields. If size of string exceeds ColumnSize string will be truncated. If size of string exceeds size of column in DB insert/update will be failed.
- ConnectionString – ODBC connection string. The attribute is required.

ODBCHistory supports the following types of fields:

- Int
- Float
- Numeric. Additional required attributes for this type are Precision (max number of decimal digits, must have value from 1 to 38) and Scale (max number of decimal digits that can be stored to the right of decimal point, should be not less than 0 and not greater that Precision)
- Date
- DateTime
- String – **default** field type

Xml-elements <KeyField> and <Field> are used to define the field in the table that will store the particular tag of the FIX message.

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="Price" DataType="Numeric" Precision="20" Scale="6">44</Field>
            <Field ColumnName="SettlDate" DataType="Date" ColumnSize="10">64</Field>
                <Field ColumnName="TransactTime">ExpireDateTime</Field>
</History>
```

KeyField element

Element defines a key field in the database table and a tag of the FIX message that will be stored in it.

Attributes are used for database fields definition in ODBC History:

ColumnName	Mandatory. Name of table field
DataType	Mandatory. Type of table field Valid values: "Int" | "Float" | "Numeric" | "Date" | "DateTime" | "String" – default value
ColumnSize	Optional
DecimalDigits	Optional, required if DataType = "Float"
Precision	Optional, required if DataType = "Numeric". Max number of decimal digits, must have value from 1 to 38
Scale	Optional, required if DataType = "Numeric". max number of decimal digits that can be stored to the right of decimal point, should be not less than 0 and not greater that Precision.

For example:

```
<KeyField ColumnSize="7" ColumnName="ClOrdID">11</KeyField>
```

Field element

Element defines the field and the tag of the FIX message that will be stored in it.

Attributes are used for database fields definition in ODBC History:

ColumnName	Mandatory. Name of table field
DataType	Mandatory. Type of table field Valid values: "Int" | "Float" | "Numeric" | "Date" | "DateTime" | "String" – default value
ColumnSize	Optional
DecimalDigits	Optional, required if DataType = "Float"
Precision	Optional, required if DataType = "Numeric". Max number of decimal digits, must have value from 1 to 38
Scale	Optional, required if DataType = "Numeric". max number of decimal digits that can be stored to the right of decimal point, should be not less than 0 and not greater that Precision.

For example:

```
<Field ColumnSize="15" ColumnName="ExDestination">100</Field>
<Field ColumnSize="19" ColumnName="Quantity" DecimalDigits="10" DataType="Float">53</Field>
<Field ColumnSize="12" ColumnName="TransactTime" DataType="Datetime">60</Field>
<Field ColumnSize="19" ColumnName="OfferPx" DecimalDigits="7" DataType="Float">133</Field>
<Field ColumnSize="19" ColumnName="OfferSize" DecimalDigits="7" DataType="Float">135</Field>
```

File History

Each File History defines regulations of storage FIX message tags within the particular file. 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.

Both of xml-elements <KeyFields> and <Fields> define set of the tags. Tags within <KeyFields> are used as a record composite key.

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>
```

KeyFields element

Contains a set of comma-separated FIX message tags. Tags are used for search the message data within the file or memory.

Fields element

Contains a set of comma-separated FIX message tags.

Memory History

Memory History defines regulations of storage FIX message tags within the FIXEdge server memory. MemoryHistory does not require additional attributes

Both of xml-elements <KeyFields> and <Fields> define set of the tags. Tags within <KeyFields> are used as a record composite key.

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>
```

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 details about Javascript and XSTL scripting here: BL Scripting with JavaScript and BL Scripting with XSLT

In case of the JavaScript usage you can control the number of parallel scripts using the *BusinessLayer.JS.ExecuteInParallel* parameter. More detailed information about it is available here: FIX Engine parameters#BusinessLayer.JS.ExecuteInParallel

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
// - 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);
```

Refer to BL Scripting with JavaScript for details.

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, isGroupValid, 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.

Refer to BL Scripting with XSLT for details.

Default routing strategies

There are two forms of default routing:

- DeliverToStrategy
- OrderFlowStrategy

DeliverToStrategy

<DeliverToStrategy> element specifies the routing destination that has been specified beforehand inside the source FIX message.

The following properties of the element are required:

- **Name** – contains the name of element to identify in factory rules.
- **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.

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

⚠ 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

<OrderFlowStrategy> element defines the orders flow to destination that can be found in the history of orders.

The following properties of element are required:

- **Name** – contains the name of element to identify in factory rules.
- **History** – contains the name of History element, where the order flow strategy will refer to specify the routing destination.
- **DestinationField** – contains a tag number with destination. For example, tag 100 <ExDestination>.

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

⚠ **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’ value should be specified in strategy description (default 100)

Below is the example of appropriate history definition:

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

Examples of routing strategies usage

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>
                <FixSession SenderCompID=".*" TargetCompID=".*"/>
        </Source>
        <Condition>
                <MatchField Tag="35" Value="D|AB|G|AC|8|9" />
        </Condition>
        <Action>
                <StrategySend Name="OrderFlowStrategy100" />
        </Action>
</Rule>
```

CME_OrderRule.xml:

```
<Rule Enabled="false" Description="Override for new order factory rule, when send to CME">
        <Source>
                <FixSession SenderCompID=".*" TargetCompID=".*"/>
        </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>
                <FixSession SenderCompID=".*" TargetCompID=".*"/>
        </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>
                <FixSession SenderCompID=".*" TargetCompID=".*"/>
        </Source>
        <Condition>
                <MatchField Tag="35" Value="c|d" />
        </Condition>
        <Action>
                <StrategySend Name="DeliverToStrategy115128" />
        </Action>
</Rule>
```

Examples of usage

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
<?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.

```
<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:

a. Message is duplicated: SrcMsg DupMsg
b. For DupMsg: Field 57 is set to "TargetSubID"
c. DupMsg is sent to the defined FIX Session ID
d. Message is duplicated: DupMsg DupDupMsg
e. For DupDupMsg: Field 57 is set to "AdminTargetSubID"
f. DupDupMsg is sent to the defined FIX Session ID

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>
```