How to process FIX messages with unexpected structure

- Recommendations
- Examples
  - 1. Update Text(58) for all message types if it is applicable
  - 2. Save the various message ids and information about parties to the database

Recommendations

The counterparty can send a FIX message with an unexpected message structure. It is possible if FIX Specification for the counterparty is unknown or the FIX dictionary is incorrect.

FIXEdge allows handling such cases if a user would explicitly check if tags are corresponding to the FIX message structure defined in the FIX dictionaries.

By default, scripts fail when FIXEdge try to access the unexpected or undefined tags so checking if tags are supported in the messages makes logic be more stable.

This approach allows keeping Business logic scripts unchanged while dictionaries are being modified or apply a single script for multiple message types with the different structure

If the user doesn't know if a tag is defined in the dictionary at the moment but there is a need to create a script using this tag the recommendations are the following:

1. Check if tag is defined for the message with functions isSupportedField(<fieldTag>), isSupportedField(<group handle> , <fieldTag>) before accessing the fields.
2. In case of success perform the required operations.

Examples

1. Update Text(58) for all message types if it is applicable

The case is useful when the user wants to mark a message which is passing certain steps in the business logic the specific rule.

The tag Text (58) doesn't present in all of the messages so the following Business logic will update it everywhere when it is applicable

Business logic configuration:

- For all messages run the UpdateText.js script

BL_Config.xml

```
        <Rule Description="Update Text for all messages from all sessions">
            <Source Name=".*" />
            <Action>
                                <!-- ... some business logic ... -->
                                <Script Language="JavaScript" FileName="../FIXEdge1/conf/UpdateText.js"/>
                                <!-- ... some business logic ... -->
            </Action>
        </Rule>
```

- The script checks and updates the value of tag 58.

UpdateText.js

```
txt = "UpdateText.js was applied to the message";
if (isSupportedField(58)) // If it is possible to update field 58
        setStringField(58, txt); // update the text
```

2. Save the various message ids and information about parties to the database

The case is useful if a user needs to save a message reference and information about parties for all possible messages passing through the FIXEdge

The example below saves information about

- Session specific information (tags: MsgSeqNum (34), MsgType (35), SenderCompID (49), TargetCompID (56), SendingTime (52), TransactTime (60))
- the message (order) type (tags: OrdStatus (39), OrdType (40), ExecType (150), SecurityID (48), Symbol (55))
- Parties data (tags: PartyID (448) , PartyIDSource (447) , PartyRole (452), PartyRoleQualifier (2376))
- And various ids (tags: ClOrdID (11), ExecID (17), ExecRefID (19), OrigClOrdID (41), SecondaryClOrdID (526), SecondaryExecID (527), ClOrdLinkID (583), ComplianceID (376))

For this example every party entry would be saved as a single row in the table.

For example, one message with 3 parties would be split and stored as 3 rows in the table.

Business logic configuration:

- Define database table as ODBC History for FIXEdge

BL_Config.xml

```
        <History Name="FIXMessages"
             StorageType="ODBC"
             TableName="FIXMessages"
             ColumnSize="64"
             ConnectionString="DSN=EPAM_TEST_DB;UID=user;Pwd=password">
      <KeyField ColumnName="SeqNum">34</KeyField>
      <KeyField ColumnName="MsgType">35</KeyField>
      <KeyField ColumnName="SenderCompId">49</KeyField>
      <KeyField ColumnName="TargetCompId">56</KeyField>
      <KeyField ColumnName="SendingTime">52</KeyField>
      <Field ColumnName="TransactTime">60</Field >

      <Field ColumnName="OrdStatus">39</Field>
      <Field ColumnName="OrdType">40</Field>
      <Field ColumnName="ExecType">150</Field>
      <Field ColumnName="SecurityId">48</Field>
      <Field ColumnName="Symbol">55</Field>

      <Field ColumnName="ClOrdId">11</Field>
      <Field ColumnName="ExecId">17</Field>
      <Field ColumnName="ExecRefId">19</Field>
      <Field ColumnName="OrigClOrdId">41</Field>
      <Field ColumnName="SecondaryClOrdID">526</Field>
      <Field ColumnName="SecondaryExecID">527</Field>
      <Field ColumnName="ClOrdLinkID">583</Field>
      <Field ColumnName="ComplianceID">376</Field>

      <Field ColumnName="PartyEntry">10453</Field>
      <Field ColumnName="PartyID">448</Field>
      <Field ColumnName="PartyIDSource">447</Field>
      <Field ColumnName="PartyRole">452</Field>
      <Field ColumnName="PartyRoleQualifier">2376</Field>

      <Field ColumnName="FullMessage">10000</Field>
    </History>
```

- Call **SaveData.js** script for all message types

BL_Config.xml

```
        <Rule Description="Save Parties and IDs for all messages from all sessions">
            <Source Name=".*" />
            <Action>
                            <!-- ... some business logic ... -->
                            <Script Language="JavaScript" FileName="../FIXEdge1/conf/SaveData.js"/>
                            <!-- ... some business logic ... -->
            </Action>
        </Rule>
```

- Script collect all data and for every entry in the repeating group saves data to the database.

SaveData.js

```
fullMessage = serializeMessage("|"); // get full message with '|' as delimiter for the reference

keyFields = [34, 35, 49, 56, 52]; // tags used as keys
commonFields = [60, 39, 40, 150, 48, 55, 11, 17, 19, 41, 526, 527, 583, 376]; // other fields
partyFields = [448, 447, 452, 2376]; // fields from the parties repeating group

// the function checks if the tag is avaliable and return the tag value or empty string.
function getValueOrEmpty (tag) {
        if(isSupportedField(tag))
        {
                result = getStringField(tag);
                return (result != null) ? result : "" ; // returns tag value or empty string.
        }
        else
                return "";
}

function getValueOrEmptyForGroup(groupHandle, entry, tag) {
        if(isSupportedField(groupHandle, entry, tag))
        {
                result = getStringField(groupHandle, entry, tag);
                return (result != null) ? result : "" ; // returns tag value or empty string.
        }
        else
                return "";
}

// prepare keys for the ODBC history
keys = [];
for (i=0; i<keyFields.length; i++)
{
        keys.push(getStringField(keyFields[i]));
}

commons = [];
for (i=0; i<commonFields.length; i++)
{
        commons.push(getValueOrEmpty(commonFields[i]));
}

if (isSupportedField(453)) {  // if party group is supported.
        noParties = getNumField(453);
        partyGrp = getGroup (453);
        if ( (noParties != null) && (noParties > 0) ) {
                for (j=0; j<noParties; j++)
                {
                        commonParty = [ j ]; // add entry
                        for (i=0; i<partyFields.length; i++)
                        {
                                commonParty.push(getValueOrEmptyForGroup(partyGrp, j, partyFields[i]));
                        }
                        commons = commons.concat(commonParty).concat(fullMessage);
                        saveToHistory("FIXMessages", keys, commons, ""); // save data per each party
                }
        } else {
                commonParty = [""]; // entry is empty
                for (i=0; i<partyFields.length; i++)
                {
                        commonParty.push("");
                }
                commons = commons.concat(commonParty).concat(fullMessage);
                saveToHistory("FIXMessages", keys, commons, "");
        }
}
else { // party group is not exist in the message so all tags should be empty
        commonParty = [""]; // entry is empty
```

```
        for (i=0; i<partyFields.length; i++)
        {
                commonParty.push("");
        }
        commons = commons.concat(commonParty).concat(fullMessage);
        saveToHistory("FIXMessages", keys, commons, "");
}
```