BL Scripting with JavaScript

Introduction

FIXEdge offers a flexible solution for Business Rules purposed to provide routing, transferring, and data manipulations for FIX messages that go through the Business Layer of FIXEdge. 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. To use this feature properly, scripts should be 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 time spent on business logic implementation
- Flexibility and simplicity of modification
- Does not bring a significant performance breakdown

FIXEdge uses SpiderMonkey JavaScript engine

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 do the proper transformations. This structure facilitates users with all the standard Javascript features and functionality. In addition to this, it enables users to use functions that provide access to the FIX message inside data and allows manipulations with fields and groups as well as whole FIX messages.

JS FIX functionality

A trivial example of a complete JS script for a business rule is shown 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
//CC - 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 - script execution will be terminated and failed. There is no way to handle syntax errors in script during script execution. User able to stop script execution using throw statement, the result in this case will be the same. User must be able to verify and handle logical errors using verification functions (like isNaN, isMessageValid(), etc) in condition statements.

Logging in JavaScript

print(<text>)

Example Code	Execution of this line results in the following record in the application log:
print (getCurrentDateStr(DATETIMEUtc));	[NOTE] 20131023-08:28:53.651 [4320] [../FixEdge/conf/test.js] - JavaScript '../FixEdge/conf/test.js' **output**: 20131023-08:28:53

Operations with session

startSession(<sessionName>)

Starts session "sessionName"

Example Code	Starts the "StartByEvent" session
`startSession("StartByEvent")`	

startSession(<senderCompId>,<targetCompId>)

Starts session where SenderCompID = "senderCompID" and TargetCompID = "targetCompID"

Example Code	Starts a session where SenderCompID = "FIXEdge" and TargetCompID = "Simple Client"
`startSession("FIXEdge","SimpleClient")`	

startSession(<senderCompId>,<targetCompId>, <sessionQualifier>) Starts session where SenderCompID = "senderCompID", TargetCompID = "targetCompID",SessionQualifierValue = "sessionQualifier"	
Example Code `startSession("FIXEdge","SimpleClient","Q1")`	Starts a session where SenderCompID = "FIXEdge", TargetCompID = "SimpleClient", SessionQualifierValue = "Q1"
disconnectSession(<sessionName>,<reason>) Disconnects session "sessionName" with reason "reason"	
Example Code `disconnectSession("DisconnectByEvent","Disconnect by message")`	Disconnects the "DisconnectByEvent"session with the "Disconnect by message" reason
disconnectSession(<senderCompId>,<targetCompId>,<reason>) Disconnects session where SenderCompID = "senderCompId" and TargetCompID = "targetCompId" with reason "reason"	
Example Code `disconnectSession("FIXEdge","SimpleClient","Disconnect by message")`	Disconnects a session where SenderCompID = "FIXEdge" and TargetCompID = "SimpleClient" with reason "Disconnect by message"
disconnectSession(<senderCompId>, <targetCompId>,<sessionQualifier>,<reason>) Starts session where SenderCompID = "senderCompID", TargetCompID = "targetCompID",SessionQualifierValue = "sessionQualifier" with reason "reason"	
Example Code `disconnectSession("FIXEdge","SimpleClient","Disconnect by message","Q1")`	Disconnects a session where SenderCompID = "FIXEdge", TargetCompID = "SimpleClient", SessionQualifierValue = "Q1" with reason "Disconnect by message"
terminateSession(<sessionName>,<reason>) Terminates session "sessionName" with reason "reason"	
Example Code `terminateSession("TerminateByEvent","Terminate by message")`	Terminates the "TerminateByEvent" session with the "Terminate by message" reason
terminateSession(<senderCompId>,<targetCompId>,<reason>) Terminates session where SenderCompID = "senderCompID" and TargetCompID = "targetCompID" with reason "reason"	
Example Code `terminateSession("FIXEdge","SimpleClient","Terminate by message")`	Terminates a session where SenderCompID = "FIXEdge" and TargetCompID = "SimpleClient" with reason "Terminate by message"

terminateSession(<senderCompId>, <targetCompId>,<sessionQualifier>,<reason>) Terminates session where SenderCompID = "senderCompID", TargetCompID = "targetCompID",SessionQualifierValue = "sessionQualifier"	
Example Code `terminateSession("FIXEdge","SimpleClient","Q1","Terminate by message")`	Terminates a session where SenderCompID = "FIXEdge", TargetCompID = "SimpleClient", SessionQualifierValue = "Q1" with reason "Terminate by message"

Please note:

1. The session will not be visible in FIXICC or manageable (start, disconnect, etc) from FIXICC/BL/JS/scheduler.
2. The termination and the disconnection occur with async way after rule completion. In case of fail <OnRuleFailEvent> will be called.

Operations with messages across BL rules

setCtxKV(key,value)

Example Code `setCtxKV("messageID","123")`	The function sets the value to the key to transfer information about the message between different BL rules. If the function argument's type is not a string then the function throws an exception.

getCtxKV(key,[defaultValue])

Example Code
 	
`var id = getCtxKV("messageID")`
 	
`var id = getCtxKV("messageID","0")`
	The function gets the value of the key from another BL rule.
	
	If the function argument's type is not a string then the function throws an exception.
	
	If the default value is specified and there is a key stored in the context then the function returns the value stored for the key.
	
	If the specified key is not stored in the context:
	
	• If the default value is specified then the function returns the default value.
	• If the default value is not specified then the function throws an exception.

delCtxKV(key)

Example Code `delCtxKV("messageID")`	The function deletes the value of the key. If the function's argument type is not a string then the function throws an exception.

Operations with group fields

getGroup(<fieldTag>)

Returns group handle for <fieldTag> group field

Example Code
 	
`hndl = getGroup(78);`
 	
`isGroupValid(hndl)`
	Gets a handle on group 78 and tests whether handle 78 is valid

getGroup(<parentGroupHandle>, <entry>, <fieldTag>)

Returns group handle for <fieldTag> group field from <parentGroupHandle>[<entry>] group

> ⚠ Numeration of <entry> starts with 0.

Example Code	

`hndl = getGroup(552);`
`isGroupValid(hndl);`
`hndl2 = getGroup(hndl, 0, 518);`
`isGroupValid(hndl2);`
`if(hndl2 != hndl){`
`  setNumField(552, 4);`
`}`
	Gets a handle on group 552 and tests whether the handle 552 is valid
	Gets a handle from the first record of 552 group on group 518 and tests whether the handle 518 is valid
	If the group handles are not identical, then increase the number of group 552 elements up to 4

removeField(<group handle>, <entry>, <fieldTag>)

Removes field from group field

> ⚠ Numeration of <entry> starts with 0.

Example Code	

`hndl = getGroup(78);`
`isGroupValid(hndl);`
`removeField(hndl, 0, 79);`
	Gets handle on group 78 and tests whether the handle 78 is valid
	Removes tag 79 from group 78

isGroupValid(<handle>)

Returns true or false depending on the handle value

Example Code	
`//see the example above`	

Bulk operations with group fields

> ⚠ Available since FIXEdge 6.12.0

bulkSetStringField(<group handle>, <starting entry>, <fieldTag>, <value>, [<max count>])

Populates multiple entries' specific tag with constant value, or copies values from list, depending on value type.

> ⚠ Numeration of <starting_entry> starts with 0.

Example Code 1

```
setNumField(268, 200);
hndl = getGroup(268);

bulkSetStringField(hndl, 0, 269, '0', 100);
bulkSetStringField(hndl, 100, 269, '1', 100);
```

Example Code 2

```
bid_qty_list = [10, 20, 30, 40, 50];
ask_qty_list = [11, 22, 33];

setNumField(268, bid_qty_list.length +
ask_qty_list.length);
hndl = getGroup(268);

bulkSetStringField(hndl, 0, 271, bid_qty_list);
bulkSetStringField(hndl, bid_qty_list.length,
271, ask_qty_list);
```

Example 1

Resizes group 268 (NoMDEntries) to 200

Gets a handle on group 268 (NoMDEntries)

For entries 0 - 99 set tag 269 (MDEntryType) to '0' ('Bid')

For entries 100 - 199 set tag 269 (MDEntryType) to '1' ('offer')

This is equivalent to code:

```
for (var i = 0; i < 100; i++) {
setStringField(hndl, i, 269, '0');
}

for (var i = 100; i < 200; i++) {
setStringField(hndl, i, 269, '1');
}
```

Example 2

Creates a bid_qty_list variable with 5 elements and ask_qty_list with 3 elements

Resizes group 268 (NoMDEntries) to 8 (total size of 2 lists)

Gets a handle on group 268 (NoMDEntries)

For entries 0-4, copy values from bid_qty_list, so that entry[i][271] = bid_qty_list[i]

For entries 5-7, copy values from ask_qty_list, so that entry[5+i][271] = ask_qty_list [i]

This is equivalent to code:

```
for (var i = 0; i < bid_qty_list.length; i++) {
  setStringField(hndl, i, 271, bid_qty_list[i]);
}

for (var i = 0; i < ask_qty_list.length; i++) {
  setStringField(hndl, bid_qty_list.length + i, 271,
ask_qty_list[i]);
}
```

bulkSetStringFieldMapped(<group handle>, <starting entry>, <fieldTag>, <source list>, <key> [, <max count>])

Populates multiple entries' specific tag by copying values from nested elements of source Javascript array.

⚠ Numeration of <starting entry> starts with 0.

Numeration of <source list> starts with 0.

Example Code

```
bids = [ [0.1, 10], [0.2, 20], [0.3, 30], [0.4,
40], [0.5, 50] ] ;

asks = [ [1.1, 11], [2.2, 20], [3.3, 30] ] ;

setNumField(268, bid_qty_list.length +
ask_qty_list.length);
hndl = getGroup(268);

bulkSetStringFieldMapped(hndl, 0, 270, bids, 0);
bulkSetStringFieldMapped(hndl, 0, 271, bids, 1);

bulkSetStringFieldMapped(hndl, bid_qty_list.
length, 270, bids, 0);
bulkSetStringFieldMapped(hndl, bid_qty_list.
length, 271, bids, 1);
```

For alternative source structure

```
bids = [ {"price": 0.1, "qty": 10},
         {"price": 0.2, "qty": 20},
         {"price": 0.3, "qty": 30},
         {"price": 0.4, "qty": 40},
         {"price": 0.5, "qty": 50}] ;

asks = [ {"price": 1.1, "qty": 11},
         {"price": 2.2, "qty": 22},
         {"price": 3.3, "qty": 33}] ;

setNumField(268, bid_qty_list.length +
ask_qty_list.length);
hndl = getGroup(268);

bulkSetStringFieldMapped(hndl, 0, 270, bids,
'price');
bulkSetStringFieldMapped(hndl, 0, 271, bids,
'qty');

bulkSetStringFieldMapped(hndl, bid_qty_list.
length, 270, bids, 'price');
bulkSetStringFieldMapped(hndl, bid_qty_list.
length, 271, bids, 'qty');
```

Creates a bids variable with 5 nested pairs of elements, and ask_qty_list with 3 pairs.

Resizes group 268 (NoMDEntries) to 8 (total size of 2 lists)

Gets a handle on group 268 (NoMDEntries)

For entries 0-4, copy Price(270) and Size(271) values from bids;

For entries 5-7, copy Price(270) and Size(271) values from asks.

This is equivalent to code:

```
for (var i = 0; i < bid_qty_list.length; i++) {
bid = bid_qty_list[i];
setStringField(hndl, i, 270, bid[0]);
setStringField(hndl, i, 271, bid[1]);
}

for (var i = 0; i < ask_qty_list.length; i++) {
ask = ask_qty_list[i];
setStringField(noMDEntries, bid_qty_list.length + i, 270, ask[0]);
setStringField(noMDEntries, bid_qty_list.length + i, 271, ask[1]);
}
```

For alternative source structure

```
for (var i = 0; i < bid_qty_list.length; i++) {
  bid = bid_qty_list[i];
  setStringField(hndl, i, 270, bid['price']);
  setStringField(hndl, i, 271, bid['qty']);
}

for (var i = 0; i < ask_qty_list.length; i++) {
  ask = ask_qty_list[i];
  setStringField(noMDEntries, bid_qty_list.length + i, 270,
ask['price']);
  setStringField(noMDEntries, bid_qty_list.length + i, 271,
ask['qty']);
}
```

Operations with message fields

removeField(<fieldTag>)

Removes field from message

Example Code

```
removeField(58);
```

Removes tag 58 from the message

createReject("session")

Creating reject

Example code

```
createReject("Target42sohSender42");
```

Creating reject and sending status message 35=3

isSupportedField(<fieldTag>)

Returns true if the field is defined in the dictionary, i.e. it can be presented in the message. False means the tag is not expected in this message type

Example Code:

```
if ( isSupportedField( 58 ) )
    removeField( 58 );
```

The example shows how to remove tag 58 from the message if it is defined in the dictionary

The script checks if tag 58 is defined for the message
And remove it from the message.

isSupportedField(<group handle> , <fieldTag>)

Returns true if the field is defined in the dictionary for the specific group, i.e. it can be presented in the repeating group of the message. False means the tag is not expected in this specific group for this message type

Example Code

```
partiesGrp = getGroup(448);
if ( isGroupValid(partiesGrp)
&& isSupportedField(partiesGrp,
2376) )
    removeField( partiesGrp, 0
, 2376 );
```

The example shows how to remove non required tag 2376 from the Parties group entry #0.

The script gets the repeating group 448 handle.
Then checks if the group is valid and present in the message. And if tag 2376 is defined in the dictionary.
Then removes tag 2376 from the group entry #0

getNumField(<fieldTag>)

Returns numeric value of the <fieldTag> field

Example Code

```
getNumField(78);
```

Returns a numeric value of field 78

getNumField(<group handle>, <entry>, <fieldTag>)

Returns number value of the <fieldTag> field from group <group handle>[<entry>]

⚠ Numeration of <entry> starts with 0.

Example Code

```
hndl = getGroup(552);

isGroupValid(hndl);

setNumField(552,getNumField(hndl, 0, 581));
```

Gets handle on group 552 and tests whether the handle 552 is valid

Gets a numeric value of field 581 and assigns it to field 552

getDateField(<fieldTag>, <format>)

Returns date handle to the <fieldTag> field value

Example Code

```
getDateField(64, YYYYMMDDLmd);
```

Gets a date value of the field 64 in YYYYMMDD format

getDateField(<group handle>, <entry>, <fieldTag>, <format>)

Returns date handle to the <fieldTag> field value from group <group handle>[<entry>]

> ⚠ Numeration of <entry> starts with 0.

Example Code	Gets handle on group 778 and tests whether the handle 778 is valid Gets a date value of the field 779 in the YYYYMMDD format from the first element of group 778

```
hndl = getGroup(778);

isGroupValid(hndl);

getDateField(hndl, 0, 779, YYYYMMDDLmd);
```

getStringField(<fieldTag>)

Returns string value of the <fieldTag> field

Example Code	Returns a string value of field 78

```
getStringField(78);
```

getStringField(<group handle>, <entry>, <fieldTag>)

Returns string value of the <fieldTag> field from group <group handle>[<entry>]

> ⚠ Numeration of <entry> starts with 0.

Example Code	Gets a handle on group 552 and tests whether handle 552 is valid Gets a string value of field 581 and assigns it to field 552

```
hndl = getGroup(552);

isGroupValid(hndl);

setStringField(552,getStringField(hndl, 0, 581));
```

getMStringField(<fieldTag>)

Returns array that contains multi string value of the <fieldTag> field

Example Code	Gets a multiple string value of field 18 and assigns it to the tmp variable If the number of strings in the tmp variable equals to 3, then the value of the first string is assigned to field 55 and the value of the second string is assigned to field 65

```
tmp = getMStringField(18);

if(3 == tmp.length){

setStringField(55, tmp[0]);

setStringField(65, tmp[1]);

}
```

getMStringField(<group handle>, <entry>, <fieldTag>)

Returns array that contains multi string value of the <fieldTag> field from group <group handle>[<entry>]

> ⚠ Numeration of <entry> starts with 0.

Example Code

```
hndl = getGroup(552);

isGroupValid(hndl);

tmp = getMStringField(hndl, 0, 18);
```

Gets a handle on group 552 and tests whether handle 552 is valid

Gets a multiple string value of field 18 found in the first entry of group 552 and assigns it to the tmp variable

getRawDataField(<fieldTag>)

returns array that contains raw data value of the <fieldTag> field

Example Code

```
tmp = getRawDataField(91);
```

Gets a raw data value of field 91 and assigns it to the tmp variable

getRawDataField(<group handle>, <entry>, <fieldTag>)

Returns array that contains raw data value of the <fieldTag> field from group <group handle>[<entry>]

⚠ Numeration of <entry> starts with 0.

Example Code

```
hndl = getGroup(5521);

isGroupValid(hndl);

tmp = getRawDataField(hndl, 0, 91);
```

Gets a handle on group 5521 and tests whether handle 5521 is valid

Gets a raw data value of field 91 found in the first entry of group 5521 and assigns it to the tmp variable

setNumField(<fieldTag>, <new value>)

assigns value <new value> to the <fieldTag> field as a NUMBER

Example Code

```
//see the sample for the getNumField();
```

setDateField(<fieldTag>, <format>, <new value>)

Assigns value <new value> to the <fieldTag> field as a DATE

Example Code

```
tmp = getDateField(64, YYYYMMDDLmd);

setYear(tmp, 1979);

setDateField(64, YYYYMMDDLmd, tmp);
```

Gets a date value of field 64 in the YYYYMMDD format and assigns it to the tmp variable

Sets the year of the date to 1979 in the tmp variable

Assigns a date from the tmp variable back to field 64

setStringField(<fieldTag>, <new value>)

Assigns value <new value> to the <fieldTag> field as a STRING

Example Code

```
// see the example for getMStringField();
```

setMStringField(<fieldTag>, <new value>)

Assigns value <new value> to the <fieldTag> field as a MString

Example Code	Sets an array of strings named tmp
`tmp = ["aa", "abb", "bbbc"];` `setMStringField(18, tmp);`	Assigns the array of strings to multiple string field 18

setRawDataField(<fieldTag>, <new value>)

Assigns value <new value> to the <fieldTag> field as a RawData

Example Code	Sets an array of numbers named tmp
`tmp = [12, 56, 0, 34];` `setRawDataField(91, tmp);`	Assigns the array of numbers to raw data field 91

setNumField(<group handle>, <entry>, <fieldTag>, <new value>)

Assigns value <new value> to the <fieldTag> field from group <group handle>[<entry>] as a NUMBER

> ⚠ Numeration of <entry> starts with 0.

Example Code	**Explanations**

`hndl = getGroup(552);`
`isGroupValid(hndl);`
`setNumField(hndl, 1, 519, 4);`
	Gets a handle on group 552 and tests whether handle 552 is valid
	Field 519 from the second entry of group 552 is assigned to 4

setDateField(<group handle>, <entry>, <fieldTag>, <format>, <new value>)

Assigns value <new value> to the <fieldTag> field from group <group handle>[<entry>] as a DATE

> ⚠ Numeration of <entry> starts with 0.

Example Code	Gets a date in the YYYYMMDD format from field 64 and assigns the obtained value to the tmp variable Gets a handle on group 5521 and tests whether handle 5521 is valid Field 64 from the first entry of group 5521 is assigned the value from the tmp variable

`tmp = getDateField(64, YYYYMMDDLmd);`
`hndl = getGroup(5521);`
`isGroupValid(hndl);`
`setDateField(hndl, 0, 64, YYYYMMDDLmd, tmp);`	

setStringField(<group handle>, <entry>, <fieldTag>, <new value>)

Assigns value <new value> to the <fieldTag> field from group <group handle>[<entry>] as a STRING

> ⚠ Numeration of <entry> starts with 0.

Example Code	

`tmp = getStringField(581);`
 	
`hndl = getGroup(5521);`
`isGroupValid(hndl);`
`setDateField(hndl, 0, 581, tmp);`
	Gets a string value from field 581 and assigns it to the tmp variable
	Gets a handle on group 5521 and tests whether handle 5521 is valid
	Field 581 from the first entry of group 5521 is assigned the value from the tmp variable

setMStringField(<group handle>, <entry>, <fieldTag>, <new value>)

Assigns value <new value> to the <fieldTag> field from group <group handle>[<entry>] as a MString

> ⚠ Numeration of <entry> starts with 0.

Example Code	

`tmp = getMStringField(18);`
 	
`hndl = getGroup(5521);`
`isGroupValid(hndl);`
`setMStringField(hndl, 0, 18, tmp);`
	Gets a multiple strings value from field 18 and assigns it to the tmp variable
	Gets a handle on group 5521 and tests whether handle 5521 is valid
	Field 18 from the first entry of group 5521 is assigned the value from the tmp variable

setRawDataField(<group handle>, <entry>, <fieldTag>, <new value>)

Assigns value <new value> to the <fieldTag> field from group <group handle>[<entry>] as a RawData

> ⚠ Numeration of <entry> starts with 0.

Example Code	

`tmp = getRawDataField(91);`
 	
`hndl = getGroup(5521);`
`isGroupValid(hndl);`
`setRawDataField(hndl, 0, 91, tmp);`
	Gets a multiple raw data value from field 91 and assigns it to the tmp variable
	Gets a handle on group 5521 and tests whether handle 5521 is valid
	Field 91 from the first entry of group 5521 is assigned the value from the tmp variable

swapFields(<fieldTag1>, <fieldTag2>)

Exchanges values of fields <fieldTag1> and <fieldTag2>

Example Code	
`swapFields(63, 573);`	Values of field 63 and field 573 were exchanged

swapFields(<group handle field 1>, <entry field 1>, <fieldTag1>, <fieldTag2>)

Exchanges values of fields <fieldTag1> (from group <group handle field 1>[<entry field 1>]) and <fieldTag2> (which can be both from group <group handle field 1>[<entry field 1>] and outside of it)

⚠ Numeration of <entry field 1> starts with 0.

Example Code	Gets a handle on group 552 and tests whether handle 552 is valid

`hndl = getGroup(552);`
`isGroupValid(hndl);`
`swapFields(hndl, 1, 11, 572);]`
	Exchanges values of ***field 11*** in the second entry of group 552 ***and field 572*** outside of it

swapFields(<group handle field 1>, <entry field 1>, <fieldTag1>, <group handle field 2>, <entry field 2>, <fieldTag2>)

Exchanges values of fields <fieldTag1> (from group <group handle field 1>[<entry field 1>]) and <fieldTag2> (from group <group handle field 2>[<entry field 2>])

⚠ Numeration of <entry field 1> and <entry field 2> starts with 0.

Example Code	Gets a handle on group 552 and tests whether handle 552 is valid

`hndl = getGroup(552);`
`isGroupValid(hndl);`
`swapFields(hndl, 1, 11, hndl, 1, 37);`
	Exchanges values of field 11 in the second entry and field 37 in the first entry of group 552

Additional Date operations

Create date from current date or using passed arguments.

getCurrentDate(<format>)

Returns handle that encapsulates current date in format <format>

Example Code	Creates the date data of the DATETIMEUtc format
`getCurrentDate(DATETIMEUtc);`	

createDate(<format>, <val1>, <val2>, ...<valn>)

Returns handle that encapsulates date in format <format> constructed using values <val1>, <val2>, ...<valn>

Example Code	Creates a date of the YYYYMMDD format with corresponding values and assigns it to the tmp variable Converts the value of the tmp variable to string and assigns this string to be the value of field 55
`tmp = createDate(YYYYMMDD, 1995, 12, 23);` `setStringField(55, dateToString (tmp));`	

getCurrentDateStr(<format>)

Returns current date in format <format>

Example Code `setStringField(55, getCurrentDateStr(DATETIMEUtc));`	Gets a current date as a string and assigns this string to be the value of field 55

createDateStr(<format>, <val1>, <val2>, ...<valn>)

Returns date in format <format> constructed using values <val1>, <val2>, ...<valn>

Example Code `tmp = createDateStr(YYYYMMDD, 1995, 12, 23);]`	Creates a date of YYYYMMDD format with corresponding values returns this date as string and assigns it to the tmp variable

Convert from date to string

dateToString(<date handle>)

Returns date in format that was mentioned in getCurrentDate() or createDate() call

Example Code `//see the example of createDate()`	

dateToString(<date handle>, <format>)

Returns date in format <format>

Example Code
 	
`tmp = createDate(YYYYMMDD, 1995, 12, 23);`
 	
`setStringField(55, dateToString (tmp, YYYYMM));`
	Creates a date from given data in one format and assigns it to the tmp variable
	
	Field 55 is assigned a value of the tmp varible

Get part of the date - year, month, day, etc.

getYear(<date handle>)

Returns the year of date <date handle>

Example Code
 	
`tmp = getCurrentDate (DATETIMEUtc);`
 	
`tmp2 = getYear(tmp);`
 	
`setDateField(64, YYYYMMDDLmd, tmp2);`
	Creates a date handle of the current date
	
	Extracts the year from the tmp variable and assigns it to the tmp2 variable
	
	Sets the value of field 64 to the beginning of current year

getMonth(<date handle>)

Returns the month of date <date handle>

Example Code
 	
`tmp = getCurrentDate (DATETIMEUtc);`
 	
`tmp3 = getMonth(tmp);`
	Creates a date handle of the current date
	
	Extracts the month value from the tmp variable and assigns it to the tmp3 variable

getWeek(<date handle>)

Returns weekof date <date handle>

Example code
 	
tmp = createDate(YYYYMMWW, 1995, 12, 2);
 	
setStringField(55, getWeek(tmp));
	Creates a date handle <date handle>
	
	Sets the value of the tmp variable to field 55

getDay(<date handle>)

Returns day of date <date handle>

Example Code
 	
`tmp = getCurrentDate (DATETIMEUtc);`
 	
`tmp4 = getDay(tmp);`
	Creates a date handle of the current date
	
	Extracts the day value from the tmp variable and assigns it to the tmp4 variable

getHour(<date handle>)

Returns hour of date <date handle>

Example Code
 	
`tmp = getCurrentDate (DATETIMEUtc);`
 	
`tmp5 = getHour(tmp);`
	Creates a date handle of the current date
	
	Extracts the hour value from the tmp variable and assigns it to the tmp5 variable

getMinutes(<date handle>)

Returns minutes of date <date handle>

Example Code
 	
`tmp = getCurrentDate (DATETIMEUtc);`
 	
`tmp6 = getMinutes(tmp);`
	Creates a date handle of the current date
	
	Extracts the minutes value from the tmp variable and assigns it to the tmp6 variable

getSec(<date handle>)

Returns seconds of date <date handle>

Example Code
 	
`tmp = getCurrentDate (DATETIMEUtc);`
 	
`tmp7 = get Sec(tmp);`
	Creates a date handle of the current date
	
	Extracts the seconds value from tmp variable and assigns it to the variable tmp7

getMSec(<date handle>)

Returns milliseconds of date <date handle>

Example Code
 	
`tmp = getCurrentDate (DATETIMEUtc);`
 	
`tmp7 = getMSec(tmp);`
	Creates a date handle of the current date
	
	Extracts the mile seconds value from the tmp variable and assigns it to the tmp7 variable

getNSec(<date handle>)

Returns nanoseconds of date <date handle>. Function is available since **FIXEdge 6.2.0**.

Example Code
 	
`tmp = getCurrentDate (DATETIMENanoUtc);`
 	
`tmp7 = getNSec(tmp);`
	Creates a date handle of the current date
	
	Extracts the nanoseconds value from the tmp variable and assigns it to the tmp7 variable

Set date part - year, month, day, etc.

setYear(<date handle>, <new val>)

Assign new value <new val> to year of date <date handle>

> **Validation rule**
>
> New value must be greater than 1970 and less than 2038

Example Code	

`tmp = getDateField(64, YYYYMMDDLmd);`
 	
`setYear(tmp, 1979);`
 	
`setDateField(64, YYYYMMDDLmd, tmp);`
	Gets the date value of field 64 in the YYYYMMDD format and assigns it to the tmp variable
	
	Sets the year of the date to 1979 in the tmp variable
	
	Assigns the date from the tmp variable back to field 64

setMonth(<date handle>, <new val>)

Assign new value <new val> to month of date <date handle>

> **Validation rule**
>
> New value must be greater than or equal 1 and less than or equal 12

Example Code	

`tmp = getDateField(64, YYYYMMDDLmd);`
 	
`setMonth(tmp, 06);`
 	
`setDateField(64, YYYYMMDDLmd, tmp);`
	Gets the date value of field 64 in the YYYYMMDD format and assigns it to the tmp variable
	
	Sets the month of the date to 06 in the tmp variable
	
	Assigns the date from the tmp variable back to field 64

setWeek(<date handle>, <new val>)

Assign new value <new val> to week of date <date handle>

> **Validation rule**
>
> New value must be greater than or equal 1 and less than or equal 5

Example Code	

tmp = createDate(YYYYMMWW, 1995, 12, 2);
 	
setWeek(tmp, 4);
 	
setStringField(55, dateToString(tmp));
	Gets the date value of field 64 in the YYYYMMWW format and assigns it to the tmp variable
	
	Sets the week of the date to 4 in the tmp variable
	
	Assigns the date from the tmp variable to field 55 in the string format

setDay(<date handle>, <new val>)

Assign new value <new val> to day of date <date handle>

> **Validation rule**
>
> New value must be greater than or equal 1 and less than or equal 31

Example Code	

`tmp = getDateField(64, YYYYMMDDLmd);`
 	
`setDay(tmp, 27);`
 	
`setDateField(64, YYYYMMDDLmd, tmp);`
	Gets the date value of field 64 in the YYYYMMDD format and assigns it to the tmp variable
	
	Sets the day of the date to 27 in the tmp variable
	
	Assigns the date from the tmp variable back to field 64

setHour(<date handle>, <new val>)

Assign new value <new val> to hour of date <date handle>

> ⚠ **Validation rule**
>
> New value must be greater than or equal 0 and less than 24

Example Code	

`tmp = getDateField(64, DATETIMEUtc);`
 	
`setHour(tmp, 10);`
 	
`setDateField(64, DATETIMEUtc, tmp);`
	Gets the date value of field 64 in the DATETIMEUtc format and assigns it to the tmp variable
	
	Sets the hour of the date to 10 in the tmp variable
	
	Assigns the date from the tmp variable back to field 64

setMinutes(<date handle>, <new val>)

Assign new value <new val> to minutes of date <date handle>

> ⚠ **Validation rule**
>
> New value must be greater than or equal 0 and less than or equal 59

Example Code	

`tmp = getDateField(64, DATETIMEUtc);`
 	
`setMinutes(tmp, 20);`
 	
`setDateField(64, DATETIMEUtc, tmp);`
	Gets the date value of field 64 in the DATETIMEUtc format and assigns it to the tmp variable
	
	Sets the minutes of the date to 20 in the tmp variable
	
	Assigns the date from the tmp variable back to field 64

setSec(<date handle>, <new val>)

Assign new value <new val> to seconds of date <date handle>

> ⚠ **Validation rule**
>
> New value must be greater than or equal 0 and less than or equal 59

Example Code	

`tmp = getDateField(64, DATETIMEUtc);`
 	
`setSec(tmp, 20);`
 	
`setDateField(64, DATETIMEUtc, tmp);`
	Gets the date value of field 64 in the DATETIMEUtc format and assigns it to the tmp variable
	
	Sets the seconds of the date to 20 in the tmp variable
	
	Assigns the date from the tmp variable back to field 64

setMSec(<date handle>, <new val>)

Assign new value <new val> to milliseconds of date <date handle>

> ⚠ **Validation rule**
>
> New value must be greater than or equal 0 and less than or equal 59

Example Code	

`tmp = getDateField(64, DATETIMEUtc);`
 	
`setMSec(tmp, 50);`
 	
`setDateField(64, DATETIMEUtc, tmp);`
	Gets the date value of field 64 in the DATETIMEUtc format and assigns it to the tmp variable
	
	Sets the milliseconds of the date to 50 in the tmp variable
	
	Assigns the date from the tmp variable back to field 64

setNSec(<date handle>, <new val>)

Assign new value <new val> to nanoseconds of date <date handle>. Function is available since **FIXEdge 6.2.0**.

Example Code	

`tmp = getDateField(64, DATETIMENanoUtc);`
 	
`setNSec(tmp, 50);`
 	
`setDateField(64, DATETIMENanoUtc, tmp);`
	Gets the date value of field 64 in the DATETIMENanoUtc format and assigns it to the tmp variable
	
	Sets the nanoseconds of the date to 50 in the tmp variable
	
	Assigns the date from the tmp variable back to field 64

Change date part with automatic date recalculation

changeYear(<date handle>, <new val>)

Add value <new val> to year of date <date handle>

Example Code	

`tmp = getDateField(64, DATETIMEUtc);`
 	
`changeYear(tmp, 2);`
 	
`setDateField(64, DATETIMEUtc, tmp);`
	Gets the date value of field 64 in the DATETIMEUtc format and assigns it to the tmp variable
	
	Adds 2 years to the value of the date in the tmp variable
	
	Assigns the date from the tmp variable back to field 64

changeMonth(<date handle>, <new val>)

Add value <new val> to month of date <date handle>

Example Code	

`tmp = getDateField(64, DATETIMEUtc);`
 	
`changeMonth(tmp, 5);`
 	
`setDateField(64, DATETIMEUtc, tmp);`
	Gets the date value of field 64 in the DATETIMEUtc format and assigns it to the tmp variable
	
	Adds 5 months to the value of the date in the tmp variable
	
	Assigns the date from the tmp variable back to field 64

changeDay(<date handle>, <new val>)

Add value <new val> to day of date <date handle>

Example Code	Gets the date value of field 64 in the DATETIMEUtc format and assigns it to the tmp variable Adds 10 days to the value of the date in the tmp variable Assigns the date from the tmp variable back to field 64

```
tmp = getDateField(64, DATETIMEUtc);

changeDay(tmp, 10);

 setDateField(64, DATETIMEUtc, tmp);
```

changeHour(<date handle>, <new val>)

Add value <new val> to hour of date <date handle>

Example Code	Gets the date value of field 64 in the DATETIMEUtc format and assigns it to the tmp variable Adds 24 hours to the value of the date in the tmp variable Assigns the date from the tmp variable back to field 64

```
tmp = getDateField(64, DATETIMEUtc);

changeHour(tmp, 24);

setDateField(64, DATETIMEUtc, tmp);
```

changeMinutes(<date handle>, <new val>)

Add value <new val> to minutes of date <date handle>

Example Code	Gets the date value of field 64 in the DATETIMEUtc format and assigns it to the tmp variable Adds 600 minutes to the value of the date in the tmp variable Assigns the date from the tmp variable back to field 64

```
tmp = getDateField(64, DATETIMEUtc);

changeMinutes(tmp, 600);

 setDateField(64, DATETIMEUtc, tmp);
```

changeSec(<date handle>, <new val>)

Add value <new val> to seconds of date <date handle>

Example Code	Gets the date value of field 64 in the DATETIMEUtc format and assigns it to the tmp variable Adds 3600 seconds to the value of the date in the tmp variable Assigns the date from the tmp variable back to field 64

```
tmp = getDateField(64, DATETIMEUtc);

changeSec(tmp, 3600);

setDateField(64, DATETIMEUtc, tmp);
```

changeMSec(<date handle>, <new val>)

Assign new value <new val> to milliseconds of date <date handle>

Example Code	Gets the date value of field 64 in the DATETIMEUtc format and assigns it to the tmp variable Adds 36 milliseconds to the value of the date in the tmp variable Assigns the date from the tmp variable back to field 64

```
tmp = getDateField(64, DATETIMEUtc);

changeMSec(tmp, 36);

setDateField(64, DATETIMEUtc, tmp);
```

changeNSec(<date handle>, <new val>)

Assign new value <new val> to nanoseconds of date <date handle>. Function is available since **FIXEdge 6.2.0**.

Example Code
 	
`tmp = getDateField(64, DATETIMENanoUtc);`
 	
`changeNSec(tmp, 36);`
 	
`setDateField(64, DATETIMENanoUtc, tmp);`
	Gets the date value of field 64 in the DATETIMENanoUtc format and assigns it to the tmp variable
	
	Adds 36 nanoseconds to the value of the date in the tmp variable
	
	Assigns the date from the tmp variable back to field 64

Routing functions

getSourceClientId()

Returns Client ID of source

Example Code `clientID = getSourceClientId();`	Gets the TA client ID of the session context where script is applied **Note:** Returns NULL value, if session is not TA client

getSourceSessionId()

Returns Session ID of source

Example Code `clientID = getSourceSessionId();`	Gets the FIX session ID of the session context where script is applied **Note:** Returns NULL value, if the session is not FIX session.

getSourceSessionQualifier()

Returns the name of SessionQualifier (String) in case if SessionQualifier is available for source session

Example Code `sessionQualifier = getSourceSessionQualifier();`	Gets the SessionQualifier name of the session context where script is applied

isSessionActive(<Session Name>)

Checks is FIX Session with name <Session Name> active

Example Code
 	
`active = isSessionActive ("SOME_SESSION");`
 	
`if (active)`
 	
`print("SOME_SESSION active");`
	Checks if FIX Session with the "SOME_SESSION" name is active
	
	Prints a message

convert(<source protocol>, <target protocol>)

Convert message from <source protocol> to <target protocol>

Example Code `convert("FIX.4.4", "FIX.4.2");`	Converts message in the given script context, replaces a source FIX44 message with converted to FIX42. **Note:** please refer to BusinessLayer.dtd to see applicable names of protocols

transform(<target protocol>, <target message type>)

Transform message from current script context to<target protocol> <target message type> message

Example Code `transform("FIX.4.2", "8");`	Transforms message in the given script context, replaces a source message with transformed to FIX42 MsgType="8". **Note:** refer to BusinessLayer.dtd to see applicable names of protocols

getMsgBySeqNum(<SenderCompId>, <TargetCompId> , <SeqNum>)

Search for message in session identified as <SenderCompId> and <TargetCompId> pair logs by < SeqNum>

getMsgBySeqNum(<SenderCompId>, <TargetCompId>, <SessionQualifier>, <SeqNum>)

Search for message in session identified as <SenderCompID>/<TargetCompID>/<SessionQualifier> set logs by < SeqNum>

Example Code	
`getMsgBySeqNum("TestSender", "TestTarget", 12345);`	Searches for message with given SeqNum in the session identified by the 'sender|target' pair. **Note:** FIX Session where search occurs must be active and use persistent storage.

send(<SenderCompID>, <TargetCompID>)

Send a message into session identified as <SenderCompID> and <TargetCompID> pair.

send(<SenderCompID>, <TargetCompID>,<SessionQualifier>)

Send a message into session identified as <SenderCompID>/<TargetCompID>/<SessionQualifier> set.

send(<session source identifier>)

Send a message into session that identified by <sessions source identifier>.

Example Code	
`send("sender", "target");` `send("exchange");`	Sends a message into the session identified by the 'sender|target' pair. Sends a message into the session identified by the 'exchange' source identifier.

handler(<handler name>, <array of additional parameters>)

Route message into handler <handler name> with array of additional parameters as JSObject (optional)

Example Code	
params = [["Storage", "file.txt"], ["ClearTime", "22:00"]] `handler("CMEHandler", params);`	Routes a message in the "CMEHandler" handler and sets additional parameters for processing: Storage = file.txt, ClearTime = 22:00

processNetStatusRequest()

Response Network Status Response message ("BD") on request

Example Code	
`processNetStatusRequest();`	Replaces a Network Status Request message ("BC") with Response ("BD") in the routing

createNotification (<category>, <reason>)

Creates notification message ("C") with <category>, <reason> then puts it into the routing

Example Code	
`createNotification("E","Session is rejected");`	Creates a notification message ("C") with "Status = ERROR", "Code=201", "Reason = Session is rejected" in three entries of group (58). **Note:** <category> applies values "N" – NOTE, "W" – WARNING, "E" – ERROR, "F" - FATAL

strategySend (<strategyName>)

Processes a message by strategy that identified by <strategyName>

Example Code `strategySend("OrderFlow5");`	Processes a message by the default routing strategy with the “OrderFlow5” name. **Note:** only default routing strategies can be used for processing: deliveryToStrategy, OrderFlowStrategy and WheelOrderFlowStrategy.

strategyReject (<historyName>, <createBusinessRejectIfFailed>)

Creates reject on message that was stored in history <historyName>, or BusinessReject (MsgType = ‘j’), if <createBusinessRejectIfFailed> is ‘true’ and rejecting has fail. <createBusinessRejectIfFailed> accepts ‘false’ by default if it is absent in parameters.

Example Code `strategyReject("orders", true);`	Rejects a message that was stored by some strategy in the ‘orders’ history. Creates ‘j’ Business Reject when message rejecting has failed. **Notes:** The method is used only in the default routing. Searching in history processes by the ‘ClOrdID’ key in the original message if its type: ‘D’, ‘AB’ or by ‘OrigClOrdID’ key in original message if its type: ‘AC’, ‘F’, ‘G’. Inapplicable for other messages.

serializeMessage()

Returns the string view of the current message. SOH will be used as a delimiter.

Example Code `msg = serializeMessage();`	Returns the string view of the message where SOH will be used as a delimiter and saves it into the "msg" variable.

serializeMessage(<tagDelimiter>)

Returns the string view of the current message. A specified symbol will be used as a delimiter.

Example Code `msg = serializeMessage("|");`	Returns the string view of the message where "|" will be used as a delimiter and saves it into the "msg" variable.

parseMessage(<msg>)

Parses the specified string message and replaces the current message with the parsed one. After that, the old message will be lost. <msg> is a string view of the message. Only messages with SOH as a delimiter can be parsed.

Example Code parseMessage("8=FIX.4.4 \x019=56\x0135=b\x0149=TESTI\x0156=TESTA\x0134=14\x0152=20030204-09:25:43\x01297=0\x0110=139\x01");	Parses a message from the specified string and replaces the current message with it.

parseMessage(<msg>, <version>)

Parses the specified string message using specified FIX protocol and replaces the current message with the parsed one. After that the old message will be lost. <msg> is a string view of the message and <version> is a session version in FIXEdge.properties. Only messages with SOH as a delimiter can be parsed.

Please note: the next action in the rule will receive the replaced message!

Example Code parseMessage("8=FIX.4.4 \x019=56\x0135=b\x0149=TESTI\x0156=TE STA\x0134=14\x0152=20030204-09:25: 43\x01297=0\x0110=139\x01", "FIX44");	Parses a message from the specified string using the FIX44 protocol and replaces the current message with it. If the 1128 tag is specified in the message for parseMessage(<msg>, <version>) function, then the value of the tag has higher priority during parsing than the version value. For instance: 1128=7 and FIXT11:FIX44 and the message has 1092 tag from FIX 5.0: parseMessage("8=FIXT.1.1 \x019=114\x0135=D\x011128=7\x0149=TESTI\x0156=TESTA\x0134=2\x0152=2 0081023-17:03:05\x0111=90001008\x0155=IBM\x0154=1\x0160=20080717- 10:00:00\x0138=4000\x0140=1\x011092=0\x0110=171\x01", "FIXT11: FIX44"); - the message is parsed successfully as FIX 5.0. 1128=6 and FIXT11:FIX50 and the message has 1092 tag from FIX 5.0: parseMessage("8=FIXT.1.1 \x019=114\x0135=D\x011128=6\x0149=TESTI\x0156=TESTA\x0134=2\x0152=2 0081023-17:03:05\x0111=90001008\x0155=IBM\x0154=1\x0160=20080717- 10:00:00\x0138=4000\x0140=1\x011092=0\x0110=171\x01", "FIXT11: FIX50"); - the following error appears: 2017-09-13 11:28:01,558 UTC ERROR [JS_interpreter] 3760 C: \B2BITS\FIXEdge\FIXEdge1\conf\parseMessage.js: failed with error: Tag 1092 is not defined for this message type. Parsing stopped at column: 136 in message D with sequence number 2.

Error Handling Functions

getErrorCode()

Returns an error code (integer value) in an undelivered message event

Example Code: setStringField(58, "Error (" + getErrorCode() + ") " + getErrorText());	This example assigns the values 'error code' and 'error text' to field 58. As a result of this function, the content of tag 58 inside the FIX message could be, for example,: "Error (500) Internal server error"

getErrorText()

Returns error text (string value) in an undelivered message event

Example Code: setStringField(58, "Error (" + getErrorCode() + ") " + getErrorText());	This example assigns the values 'error code' and 'error text' to field 58. As a result of this function, the content of tag 58 inside the FIX message could be, for example,: "Error (500) Internal server error"

History functions

saveToHistory (<historyName>, <keyFields>, <commonFields>, <expireDateTime>)

Insert a new record in history with name **<historyName>**. Returns true if the function succeeded and returns false otherwise.

- **<keyFields> -** is an array of values of KeyField elements from history definition. Usually, it is a list of fields defining the primary key and should be unique across all inserts.
- **<commonFields>-** is an array of values Field elements from history definition.
- **<expireDateTime> -** String with a timestamp with an expire date-time in ODBC History. The expired fields will be erased from history during the ClearHistory procedure. An empty string means there is no expiration date for the record.

The ExpireDateTime field should be defined both in History and in the Database.

If the History was created with the expireDateTime field and the expireDateTime argument of the saveOrUpdate function is absent then the following ERROR is printed to the log on saving a new record to the History:

ERROR History '<history_name>' unable to insert: the ExpireDateTime filed is not defined!

> The ExpireDateTime field of the History is optional since FIXEdge 6.14.0 release. If the History was created without the ExpireDateTime field then all new records will be inserted in it without expireDateTime value specified in the saveToHistory function. In such a case the warning will be logged:
>
> *WARN [Save to history] <history_name>': the ExpireDateTime value is passed but the field was not defined. The value is ignored.*

For more details about the history definition please see ODBC History

Example Code:

```
key = new Array("snd","tgt");
common = new Array("ClOrdID","
dest");
saveToHistory("orders", key,
common, "20101010-10:52:00");
```

Insert a new record in the table “orders”.

The inserted data is

```
SELECT * FROM orders;
+--------------+--------------+---------+------+--------------------+
| SenderCompID | TargetCompID | ClOrdID | Text | ExpireDateTime     |
+--------------+--------------+---------+------+--------------------+
| snd          | tgt          | ClOrdID | dest |  20101010-10:52:00 |
+--------------+--------------+---------+------+--------------------+
```

updateHistory (<historyName>, <keyFields>, <commonFields>, <expireDateTime>)

Update a record in history with name **<historyName>** with Primary key defined in **<keyFields>** array. Returns true if the function succeeded and returns false otherwise.

- **<keyFields> -** is an array of values of KeyField elements from history definition. Usually, it is a list of fields defining the primary key and should be unique across all inserts.
- **<commonFields>-** is an array of values Field elements from history definition.
- **<expireDateTime> -** String with a timestamp with an expire date-time in ODBC History. The expired fields will be erased from history during the ClearHistory procedure. An empty string means there is no expiration date for the record.
  The ExpireDateTime field should be defined both in History and in the Database.

> The ExpireDateTime field of the History is optional since FIXEdge 6.14.0 release. If the History was created without the ExpireDateTime field then records in it will be updated without expireDateTime value specified in the updateHistory function. In such a case the warning will be logged:
>
> *WARN [Update history] <history_name>': the ExpireDateTime value is passed but the field was not defined. The value is ignored.*

For more details about the history definition please see ODBC History

Example Code:

```
key = new Array("snd","tgt");
common = new Array("",
"dest2");
updateHistory("orders", key,
common, "20121221-10:52:00");
```

updateToHistory function params:

The record in history with primary key SenderCompID=**snd** and TargetCompID=**tgt** in the table "**orders**" is updated.

The column '**ClOrdID**' is updated with ***NULL*** value, the column '**TEXT**' is updated with '***dest2***' value.

The ExpireDateTime column value ***'20101010-10:52:00'*** is updated with value ***'20121221-10:52:00'***.

```
SELECT * FROM orders;
+--------------+--------------+---------+-------+-------------------+
| SenderCompID | TargetCompID | ClOrdID | Text  | ExpireDateTime    |
+--------------+--------------+---------+-------+-------------------+
| snd          | tgt          | NULL    | dest2 | 20121221-10:52:00 |
+--------------+--------------+---------+-------+-------------------+
```

saveOrUpdate(<historyName>, <keyFields>, <commonFields>, <expireDateTime>)

Insert a new record in history with name **<historyName>**. If inserting fails then try to update a record in history with the name **<historyName>** with the Primary key defined in the **<keyFields>** array.

Returns **true** if any of the operations to insert or updates are successful.

Returns **false** both operations failed

- **<keyFields> -** is an array of values of KeyField elements from history definition. Usually, it is a list of fields defining the primary key and should be unique across all inserts.
- **<commonFields>-** is an array of values Field elements from history definition.
- **<expireDateTime> -** String with a timestamp with an expire date-time in ODBC History. The expired fields will be erased from history during the ClearHistory procedure. An empty string means there is no expiration date for the record.

The ExpireDateTime field should be defined both in History and in the Database.

If the History was created with the expireDateTime field and the expireDateTime argument of the saveOrUpdate function is absent then the following ERROR is printed to the log on saving a new record to the History:

ERROR History '<history_name>' unable to insert: the ExpireDateTime filed is not defined!

> The ExpireDateTime field of the History is optional since FIXEdge 6.14.0 release. If the History was created without the expireDateTime field then records in it will be saved or updated without the expireDateTime value specified in the saveOrUpdate function. In such a case the warning will be logged:
>
> *WARN [Update history/Save to history] <history_name>': the ExpireDateTime value is passed but the field was not defined. The value is ignored.*

For more details about the history definition please see ODBC History

> Insert failures will be recorded in the application log.

Example Code:

```
key = new Array("snd","tgt");
common = new Array
("ClOrdIDUpdated","
destUpdated");
saveOrUpdate("orders", key,
common, "");
```

Insert a new record in the table “**orders**”.

If the record in history with primary key SenderCompID=**snd** and TargetCompID=**tgt** exists in the table “**orders**” then the column '**ClOrdID**' is updated with a ***NULL*** value, the column '**TEXT**' is updated with '***dest2***' value for the record.

The ExpireDateTime has an empty value so the record will be not cleaned up

Sample:

```
SELECT * FROM orders;
+--------------+--------------+----------------+-------------
+----------------+
| SenderCompID | TargetCompID | ClOrdID        | Text        |
ExpireDateTime |
+--------------+--------------+----------------+-------------
+----------------+
| snd          | tgt          | ClOrdIDUpdated | destUpdated
|                |
+--------------+--------------+----------------+-------------
+----------------+
```

getFromHistory (<historyName>, <keyFields>, <columnName>)

Retrieves field value in the history <historyName>. Searching criteria is the <keyFields> for a record and <columnName> for column where value reads. Returns target field value, but if such value wasn't found, returns 'undefined'.

Example Code:

```
key = new Array("snd","tgt");
time = getFromHistory("orders",
key, "ExpireDateTime");
```

Returns the ‘20121221-10:52:00’ value from the record with key (snd,tgt) that has been updated by the previous sample.

Notes: the column which is created by the <expireDateTime> parameter always has the “ExpireDateTime” name in history.
Other columns have names as described in the history description in BL_Config.xml or as a tag number in string representation if the column name is absent in the history description.

getRecordFromHistory (<historyName>, <keyFields>)

Retrieves an array of the fields from the record with key <keyFields>, located in the history <historyName>. Returns target record, but if such record wasn't found, returns 'undefined'.

Example Code:

```
key = new Array("snd","tgt");
record = getRecordFromHistory
("orders", key);
print(record);
```

Returns a fields array of the record with the key (snd,tgt), then prints content into a log.
The result will show “snd, tgt, ClOrdID, dest2, 20121221-10:52:00” if the record (snd, tgt) has been saved and updated by previous samples.

Notes: Empty fields have NULL values and printing represents it as “<null>”.
If <keyFields> is composite, then all fields in the key set should be filled.

removeRecordFromHistory (<historyName>, <keyFields>)

Removes entire record with key <keyFields>, located in the history <historyName>. Returns a number of deleted records.

Example Code:

```
key = new Array("snd","tgt");

removeRecordFromHistory

("orders", key);
```

Removes a record with a key (snd, tgt) from the ‘orders’ history.

Notes: If <keyFields> is composite, then all fields in the key set should be filled.

removeRecordFromHistoryByCompositeKey (<historyName>, <keyFields>)

Removes entire record by matching of composite key with <keyFields>, located in the history <historyName>. Returns a number of deleted records.

Example Code:

```
key = new Array("snd","");

ndeleted =
removeRecordFromHistoryByComposit
eKey ("orders", key);
```

Removes all records with keys which have first the “snd” field in a key.

Notes: the composite <keyFields> set should contain at least one field. All empty fields in a key will be ignored in searching.

Security functions

⚠ Available since FIXEdge 5.10.1

decryptString(<text>)

Decrypts string using AES 128-bit encryption mechanism. It is useful for encrypted tags (554, 925) in Logon messages.

Example Code:	Decrypts tag 554
`encryptedPass = getStringField ("554");` `pass = decryptString (encryptedPass);`	

hashString(<text>)

Calculates SHA256 hash for the sting.

Example Code:	Getting hash for the "MyPassword" string. **Note:** string for calculating hash should be not "null".
`hash = hashString("MyPassword");`	

Predefined FIX date formats

1. ***HHMMSSUtc*** for the UTC "HH:MM:SS" date format
2. ***HHMMSSsssUtc*** or ***HHMMSSMilliUtc*** for the UTC "HH:MM:SS.sss" date format
3. ***HHMMSSMicroUtc*** for the UTC "HH:MM:SS.ssssss" date format
4. ***HHMMSSNanoUtc*** for the UTC "HH:MM:SS.sssssssss" date format
5. ***YYYYMM*** for the "YYYYMM" date format
6. ***YYYYMMDD*** for the"YYYYMMDD" date format
7. ***YYYYMMWW*** for the"YYYYMMWW" date format, where WW - number of week "w1"..."w4"
8. ***DATETIMEUtc*** for the UTC "YYYYMMDD-HH:MM:SS" date format
9. ***DATETIMEsssUtc*** or ***DATETIMEMilliUtc*** for the UTC "YYYYMMDD-HH:MM:SS.sss" date format
10. ***DATETIMEMicroUtc*** for the UTC "YYYYMMDD-HH:MM:SS.ssssss" date format
11. ***DATETIMENanoUtc*** for the UTC "YYYYMMDD-HH:MM:SS.sssssssss" date format
12. ***YYYYMMDDLmd*** for the local market "YYYYMMDD" date format