CME iLink MSGW and DropCopy 4.0 interfaces support in FA C++

CME iLink MSGW and DropCopy 4.0 support is available since FA C++ version 2.16

Key CME requirements

New session model (http://www.cmegroup.com/confluence/display/EPICSANDBOX/New+iLink+Architecture#NewiLinkArchitecture-MSGWSessionModel)

3 types of logon (http://www.cmegroup.com/confluence/display/EPICSANDBOX/Session+Layer+-+Logon):

- Begin week logon
- Mid week logon
- in-session logon

Enchanced resend sequence numbers logic (http://www.cmegroup.com/confluence/display/EPICSANDBOX/Session+Layer+-+Resend+Request)

- don't send an additional resend request if real-time messages come
- limit of 2500 messages per single resend request
- 48-hours restriction for resend request messages (for Drop Copy 4.0 only)

Fault Tolerance Indicator (http://www.cmegroup.com/confluence/display/EPICSANDBOX/Fault+Tolerance)

Encapsulated payload (http://www.cmegroup.com/confluence/display/EPICSANDBOX/Drop+Copy+4.0#DropCopy4.0-Encapsulatedpayload)

How to build CME Client sample

Platforms

CME Client sample is available for Windows Linux platforms that are supported by FIX Antenna C++..

Prerequisites

For a Windows platform installed MS Visual Studio 2010 or higher is required.

For a Linux platform installed GCC (GNU Compiler Collection) v. 4.4.7 or higher and installed Make tool are required.

How to build

Unzip the FIX Antenna package to any folder and proceed with actions below according to your platform.

On Windows navigate to the "samples/" folder and open a file "Samples.sln" in MS Visual Studio.

See the CMEClient sample in this solution.

If the build is successful you will find executable file "CMEClient.exe" at "samples/CMEClient/bin".

On Linux navigate to the "samples/" folder and execute make:

make

If the build is successful you will find executable file "CMEClient" at "samples/CMEClient/bin".

CME client sample usage

Configure

Before running the sample correct the engine.property file.

Separate FIX-dictionaries are required for CME Drop Copy 4.0 and CME iLink MSGW.

For the DropCopy 4.0 connection use

```
DictionariesFilesList = ../../../data/fixdic42.xml;../../../data/additional_DropCopy40.xml
```

but for the iLink MSGW use

```
DictionariesFilesList = ../../../data/cmeilink_fix42.xml
```

Dictionaries files are loaded automatically by the engine.

All the properties described below (with prefix CMESample) are loaded in not by engine's but by sample's code. (See class Params in Params.cpp of sample source directory).

The property

```
CMESample.IsDropCopy = true
```

controls if sample is for Drop Copy. Set it to true if connection to Drop Copy is assumed or set it to false if connection to iLink MSGW trading interface.

The properties below controls the connection parameters:

```
CMESample.Port = 9026
CMESample.SenderCompId = TSTCIDN // the last symbol of SenderCompID should be N as hot failover is not supported
CMESample.SenderSubId = TST
CMESample.SenderLocationId = RU // this parameter specifies location of client and can be set arbitrarily
CMESample.Password = password123
```

Obtain these parameters from CME.

The properties below control market segments on CME to connect:

```
CMESample.MarketSegments = 99,74
CMESample.MarketSegment.99.PrimaryHost = 69.50.112.199
CMESample.MarketSegment.99.BackupHost = 69.50.112.205

CMESample.MarketSegment.74.PrimaryHost = 127.0.0.1
CMESample.MarketSegment.74.BackupHost = 69.50.112.182
```

IP adresses of market data segments are published by CME. For more details see http://www.cmegroup.com/confluence/display/EPICSANDBOX/New+iLink+Architecture#NewiLinkArchitecture-SFTP-MSGWConfiguration

In the sample above two market segments are configured, but there is no restriction on number of them - feel free to add all market data segments.

```
CMESample.ProcessMessagesOutOfOrder = true
```

This parameter controls how application level messages should deliver to app code. Actually CME requires that application-level messages that are received out of order are taking into account by client application. For more info see http://www.cmegroup.com/confluence/display/EPICSANDBOX/Session+Layer+-+Resend+Request

If this parameter is set to true then it is garantee that messages are delivered in right order (sequence numbers are inceremented one-by one)

If it is set to false then application - level message are delivered as soon as FIXAntenna receives them from CME.

For more details see MessageDispatcher class implementation in MessageDispatcher.cpp

Run

Before running the sample make sure you have a valid license file and it is placed to the folder specified in "samples/CMEClient/bin/engine.properties" LicenseFile parameter (default value is ./../../../engine.license, that is the root folder where you unzipped the package).

Without a valid license file FIX Engine will fail to start and you won't be able to run the sample.

Run run.bat or run.sh files depending on your platform to run the sample.

Logs for the current session will be stored in the "samples/CMEClient/bin/logs" folder.

Menu:
==========
1. Begin Week Logon
2. Mid Week Logon
3. In-Session Logon
4. Disconnect session
5. Set Seq Nums
6. Switch Backup to Primary Host Ip
7. Reconnect Session Mid Week
8. Send message (for Trading interface only)
9. Send Test Request message (used in In-Session Logon)
0. Exit
==========

After the start you will also see all needed parameters, parsed from "engine.properties" file:

[INFO] 20151215-16:14:28.226 [6700] [CME Initiator] - ----------
[INFO] 20151215-16:14:28.226 [6700] [CME Initiator] - Parsed parameters (Market Segments with empty Id, Primary or Backup Hosts were skipped):
[INFO] 20151215-16:14:28.242 [6700] [CME Initiator] - ----------
[INFO] 20151215-16:14:28.242 [6700] [CME Initiator] - Port: 9026
[INFO] 20151215-16:14:28.242 [6700] [CME Initiator] - SenderCompId: TSTCIDN
[INFO] 20151215-16:14:28.257 [6700] [CME Initiator] - SenderSubId: TST
[INFO] 20151215-16:14:28.257 [6700] [CME Initiator] - SenderLocationId: RU
[INFO] 20151215-16:14:28.257 [6700] [CME Initiator] - Password: password123
[INFO] 20151215-16:14:28.257 [6700] [CME Initiator] - ApplicationSystemName: FIXAntenna
[INFO] 20151215-16:14:28.257 [6700] [CME Initiator] - TradingSystemVersion: 2.16.0
[INFO] 20151215-16:14:28.273 [6700] [CME Initiator] - ApplicationSystemVendor: B2BITS
[INFO] 20151215-16:14:28.273 [6700] [CME Initiator] - Handle out of order messages as soon as they come
[INFO] 20151215-16:14:28.273 [6700] [CME Initiator] - Assume connecting to DropCopy interface
[INFO] 20151215-16:14:28.273 [6700] [CME Initiator] - Market Segments:
[INFO] 20151215-16:14:28.288 [6700] [CME Initiator] - Id: 74
[INFO] 20151215-16:14:28.288 [6700] [CME Initiator] - Primary Host: 69.50.112.197
[INFO] 20151215-16:14:28.288 [6700] [CME Initiator] - Backup Host: 69.50.112.182
[INFO] 20151215-16:14:28.288 [6700] [CME Initiator] - Id: 99
[INFO] 20151215-16:14:28.288 [6700] [CME Initiator] - Primary Host: 69.50.112.199
[INFO] 20151215-16:14:28.304 [6700] [CME Initiator] - Backup Host: 69.50.112.205
[INFO] 20151215-16:14:28.304 [6700] [CME Initiator] - ----------

Please make sure every Market Segment is parsed correctly before continuing.

For each particular Market Segment a separate Session is created.

After the start and after each operation this menu will be displayed.

To execute an operation enter the number (0 - 9) and hit Enter.

Let's now look at each operation from the menu in more detail:

1. Begin Week Logon

Establishes connection with CME using the Beginning of Week Logon (35=A) message for a Session assigned to a particular Market Segment.

It is the very first logon message the client system sends for the week.

For more information about this type of Logon please refer to http://www.cmegroup.com/confluence/display/EPICSANDBOX/Drop+Copy+Session+Layer+-+Logon#DropCopySessionLayer-Logon-BeginningofWeekLogon

2. Mid Week Logon

Establishes connection with CME using the Mid-Week Logon (35=A) message for a Session assigned to a particular Market Segment.

It is used for any subsequent logons, after the beginning of the week.

The Mid-Week Logon uses a sequence number series that continues from the next sequence number where the client logged off or was disconnected.

As a result, the Mid-Week Logon cannot have a sequence number set to '1'. The requirements of Mid-Week Logon are similar to the Beginning of Week Logon except for the sequence number requirement.

For more information about this type of Logon please refer to http://www.cmegroup.com/confluence/display/EPICSANDBOX/Drop+Copy+Session+Layer+-+Logon#DropCopySessionLayer-Logon-Mid-WeekLogon

3. In-Session Logon

It is used to reset sequence numbers for a Session assigned to a particular Market Segment while the client system is already logged on.

In-Session Logon should only be used to recover from catastrophic failure.

Session should be established in order to apply In-Session Logon.

The client system must send a Test Request (tag 35-MsgType=1) message before sending an In-Session Logon (tag 35-MsgType=A) message. If not sent in that order, the client system may lose messages that cannot be requested again as the sequence number may be reset to '1' for both parties.

For more information about this type of Logon please refer to http://www.cmegroup.com/confluence/display/EPICSANDBOX/Drop+Copy+Session+Layer+-+Logon#DropCopySessionLayer-Logon-In-SessionLogon

4. Disconnect session

Shutdowns established connection for a Session assigned to a particular Market Segment and recreates the session without saving In and Out Sequence numbers.

After this operation Logon can be performed again.

5. Set Seq Nums

It is used to change In and Out Sequence numbers for a Session assigned to a particular Market Segment or to leave it as-is.

6. Switch Backup to Primary Host Ip

Shutdowns established connection for a Session assigned to a particular Market Segment and reconnects Session to the Backup Host IP saving In and Out Sequence numbers.

Session should be established in order to perform this operation.

Backup and Primary Hosts will be switched after this operation.

7. Reconnect Session Mid Week

Reconnects established Session assigned to a particular Market Segment saving In and Out Sequence numbers.

Operation allows user to set a different In Sequence number, received as Start Sequence Number from Session Level Reject message or leave it as-is.

It supports a 48-hours restriction for Resend Request messages and helps to avoid sending Resend Request manually after receiving Session Level Reject.

8. Send message (for Trading interface only)

Allows user to send a FIX message from a file to Session assigned to a particular Market Segment.

9. Send Test Request message (used in In-Session Logon)

Sends a Test Request message to Session assigned to a particular Market Segment.

It is recommended to be used by Client before performing In-Session Logon.

Otherwise the client system may lose messages that cannot be requested again as the sequence number may be reset to '1' for both parties.

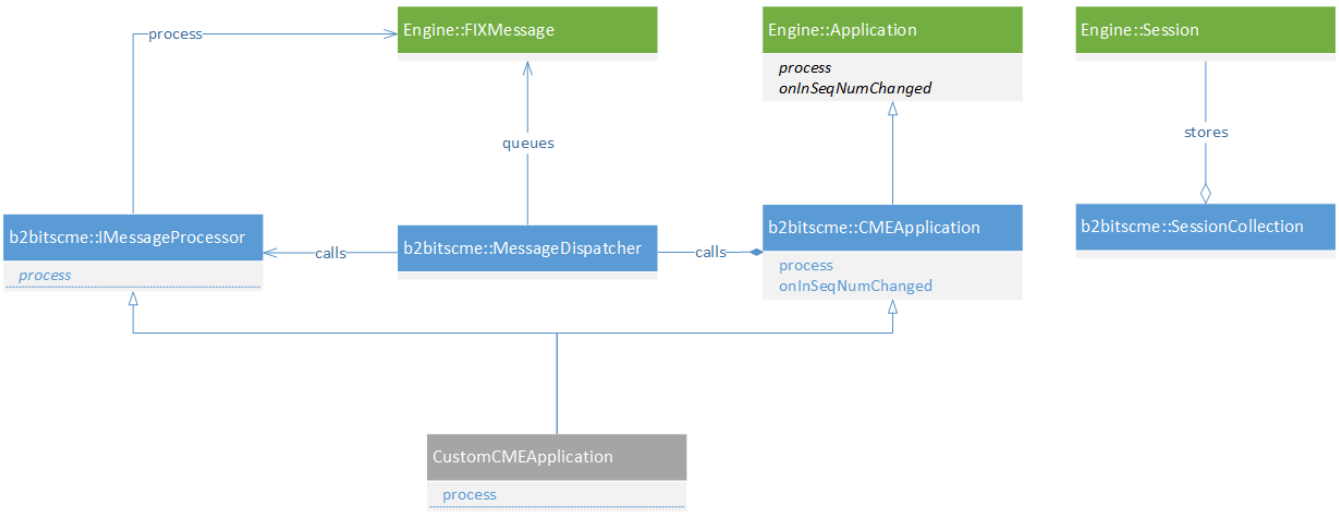
How to create an application for iLink MSGW

Samples structure

In order to understand the key features of an application for CME consider the samples' source code. It is organized so that there is a directory B2BITS_CME with several classes that are supposed to be used for creating iLink MSGW application.

These classes are shipped with source code (it means that .cpp and .h files are there). They can be modified by developer, but in most cases they can be used as-is.

The main idea of it is shown on the picture below



Green-colored: classes that are shipped within FIX Antenna's dll (a developer can't modify them) For more info on these classes please see the documenation (http://corp-web.b2bits.com/fixacpp/doc/html/)

Blue-colored: classes that are shipped as sources (a developer can modify them but it is not necessary)

Gray-colored: just a sample of user code (definitely should be modified or re-written by a developer).

So in order to start with FIX Antenna C++ it is required to create a class that inherite from b2bitscme::IMessageProcessor and from b2bitscme:: CMEApplication.

It is required to override the method

```
virtual void process(const Engine::FIXMessage& fixMsg, const Engine::Session& aSn, bool isFromQueue);
```

from the IMessageProcessor interface. All application-level mesages are passed by theconst Engine::FIXMessage& fixMsg parameter.

Any other calbacks from CMEApplication can be overrided, but it is required that parent class' methods are called for in callbacks as

```
virtual bool process(const Engine::FIXMessage& fixMsg, const Engine::Session& aSn)

virtual void onInSeqNumChanged(int oldValue, int newValue, const Engine::Session& sn)

virtual void onAfterMessageIsParsed(Engine::FIXMessage& msg, const Engine::Session& sn)
```

Differencies from the classic applications. Reasons of using special MessageDispatcher

CME applications are differ from the classical FIX Application, where just inheritance from the Engine::Apllication is assumed. For the CME application we need to use special class MessageDispatcher.

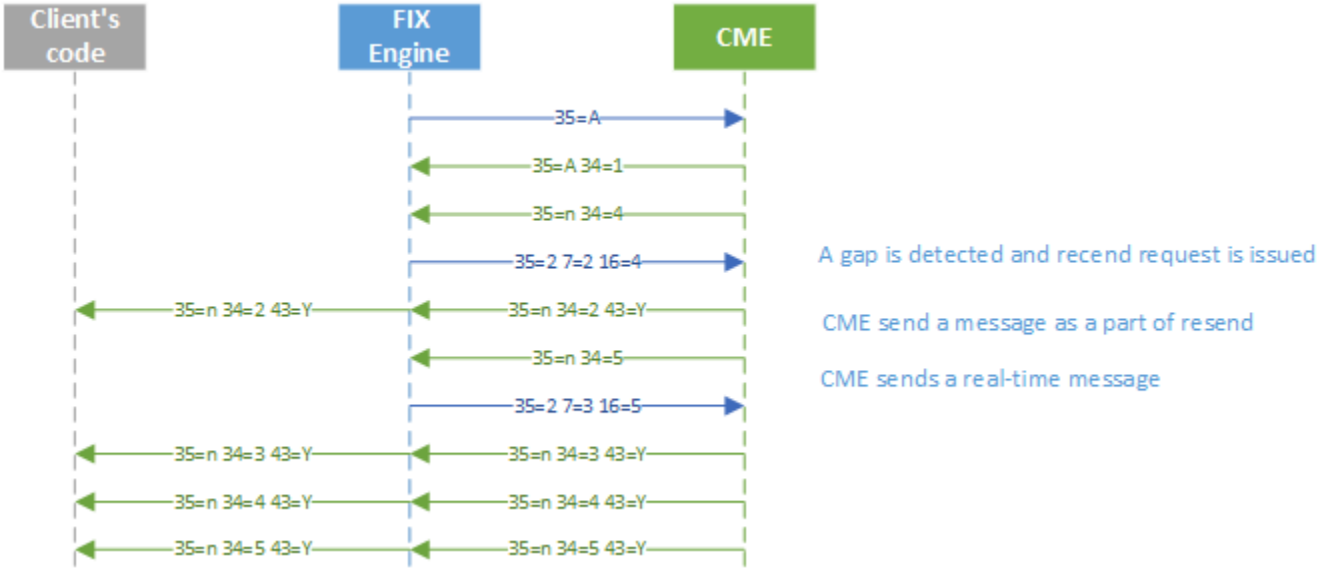
It is required for the only one reason: CME enhanced resend request logic demands that all real-time messages that are sent during resend request don't cause an additional resend request. There are two strategies that can be used with such real-time messages:

- immideately process that messages;
- store it in some queue and process once it turn comes.

Lets show these approaches with the sequence diagrams:

1. Classical behaviour (not recommended to use by CME).

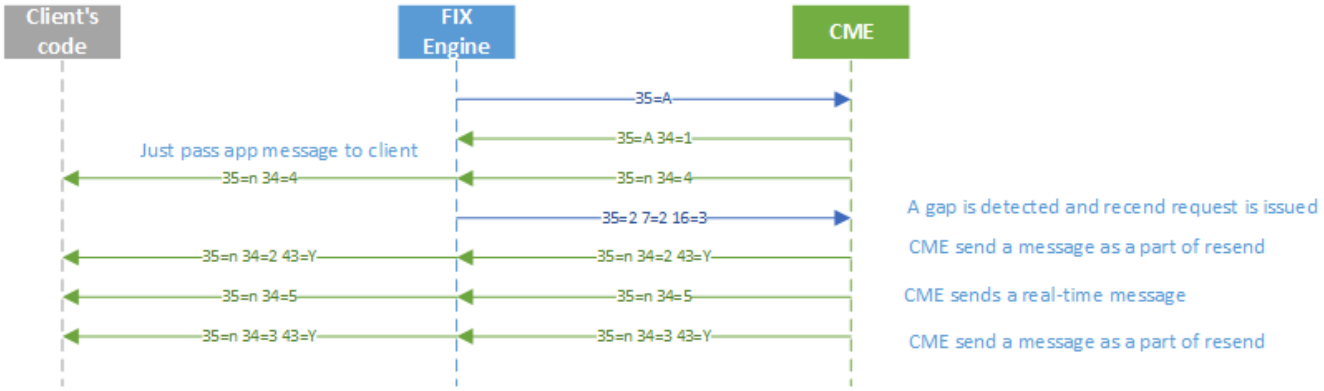
```
DeliverAppMessagesOutOfOrder = false
```



2. Pass app messages to user code as soon as they come (even if they are out of order).

```
DeliverAppMessagesOutOfOrder = true
```

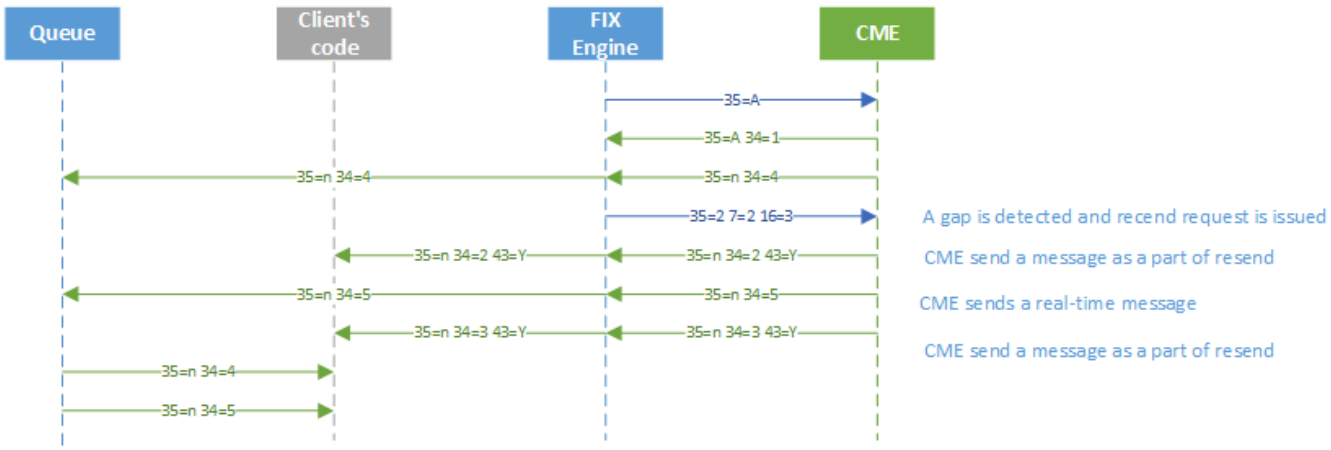
```
CMESample.ProcessMessagesOutOfOrder = true
```



3. Pass out-of order app messages to the queue. Pass them to client's code as soon as it turns comes.

```
DeliverAppMessagesOutOfOrder = true
```

```
CMESample.ProcessMessagesOutOfOrder = false
```



So the class b2btscme::MessageDispatcher is used to organize the queue for out of order messages and to deliver them to client once their turn comes. If the parameter

```
CMESample.ProcessMessagesOutOfOrder = true
```

is set to true, then MessageDispatcher just pass app-messages to client's code.

Please note, that unlike classical approach, user should recieve app messages not in the callback, inherited from

```
Engine::Application::process(const Engine::FIXMessage& fixMsg, const Engine::Session& aSn)
```

but in the callback, inherited from

```
void b2bitscme::IMessageProcessor::process(const Engine::FIXMessage& fixMsg, const Engine::Session& aSn, bool isFromQueue).
```

The first one should just delegate the call to the MessageDispatcher.

The third parameter isFromQueue can be used to determine if a message comes from a queue or directly from socket.

Also please note that as unlike CGW FIX Session MSGW Session requires separate TCP connections established (one connection per market segment), so FIX messages can come to the process callback from different threads and additional synhronization is required depending on client's application logic.

Some important tips on developing an application for CME with FIX Antenna C++

FIX Session creation

It is quite important to

- specify deliverAppMessagesOutOfOrder = true in order to enable enhanced resend request scenario
- specify resendRequestBlockSize = 2500 in order meet CME limitation of 2500 msg per single resend request
- specify targetSubID as a market segment id
- create FIX Session object using Engine::SessionId that accepts the session qualifier parameter - in order to have an ability to create more than one FIX Session with the same SenderCompID and TargetCompID pair

See the code snippet below for more details:

```
Engine::SessionExtraParameters ssnParams;

ssnParams.deliverAppMessagesOutOfOrder_ = true;

ssnParams.resendRequestBlockSize_ = 2500;

ssnParams.pTargetSubID_ = marketSegmentId;

ssnParams.pSenderSubID_ = params.senderSubID_;

...

std::string sessionQualifier = marketSegmentId;

Engine::FixEngine::singleton()->createSession(app, Engine::SessionId(params.
senderCompID_, "CME", sessionQualifier), Engine::FIX42, ssnParams);
```

For the complete code sample please see the

```
SessionCollection::createSession( const std::string &marketSegmentId, Engine::Application* app, const Params& params)
```

method.

Logon to CME

In each cases of logon to CME it is required to use "custom logon" in order to specify required by CME tags:

- 1603 - ApplicationSystemName
- 1604 - TradingSystemVersion
- 1605 - ApplicationSystemVendor
- 95 - RawDataLength (used for password length)
- 96 - RawData (used for password)

For more details see

```
std::auto_ptr<Engine::FIXMessage> createLogonMessage(const Params& params, SessionPtr session)
```

in the CMEApplication.cpp

For **Begin Week Logon** it is required that no 141 tag is used in logon message or 141=N. Also it is required to set outgoing and incoming sequence numbers to 1.

```
session->resetSeqNum(Engine::Session::RESET_SEQNUM_STRATEGY);

std::auto_ptr<Engine::FIXMessage> msg = createLogonMessage(*params_, session);
```

```
msg->set(FIXFields::ResetSeqNumFlag, "N");

session->connect(30, *msg, host, params_->port_ );
```

See CMEApplication::beginWeekLogon method for more info.

For **Mid Week Logon** it is required to keep the same sequence numbers as from previous connect. Also it is required to use 141=N or don't specify this tag in the logon message. See CMEApplication::midWeekLogon for more info.

In-session Logon to CME is using to restore from some disaster situation. It should be used while FIX Session is already in established state. So it is required to reset sequence numbers on existing session:

session->resetSeqNum(Engine::Session::RESET_SEQNUM_AND_SEND_LOGON_STRATEGY);

For more info see CMEApplication::inSessionLogon method.

CME FIX language specification

In order to fit CME FIX specification it is requred to use special fix dictionaries for FIX Antenna C++. For Trading interface please use cmeilink_fix42.xml, for DropCopy interface - ixdic42.xml among with additional_DropCopy40.xml.

These files are located in data folder of FIX Antenna's package.

Sending FIX messages to CME (for iLink MSGW trading interface)

For each sent message ManualOrderIndicator field should be set to "Y" or "N":

For more details see CMEApplication::send method

Encapsulated payload (for DropCopy 4.0 interface)

CME interface Drop Copy 4.0 sends drop copy messages in encapsulated mode (i.e. FIX over FIX approach is used). See msg sample below:

8=FIX.4.2 9=475 35=n 49=CME 56=TSTCIDN 34=4 50=G 57=PAA 143=RU 52=20151110-12:57:23.715 122=20151007-16:09:32.854 212=356 213=<RTRF>**8=FIX.4.2 9=320 35=8 34=4291 369=4267 52=20151007-16:09:32.854 49=CME 50=G 56=G86000N 57=DUMMY 143=US,IL 1=00521 6=0 11=ACP1444234172815 14=0 17=99218:12219 20=0 37=9913161431 38=1 39=0 40=2 41=0 44=9980.5 48=998902 54=1 55=90 59=0 60=20151007-16:09:32.852 107=2EJZ4 150=0 151=1 167=FUT 432=20151007 1028=Y 1091=N 9717=ACP1444234172815 10=235** </RTRF> 369=995 10=175

So message is wrapped in 213 tag of 35=n message. Note that standard FIX 4.2 protocol doesn't support "n" message type, so correct fix dictionary should be used. For more details on how extract encapsulated drop copy message please see CustomCMEApplication::parseDropCopyReport method in samples.

48-hours restriction for resend request messages

A Drop Copy 4.0 interface have the 48-hours restriction for resend request. So if a resend request is issued with tag 7 that specifies seqnum earlier than 48 hours - then a session - level reject (35=3) from CME comes. Client should send a new resend request and (7) tag of a new resend request should be get from the tag (5024) of incoming 35=3. Actually generating and handling resend requests is done in atomated mode inside Engine's code. So a client is not supposed to generate its own resend requests. In case such session-level reject comes it is recommended that FIX session is disconnected, incoming sequence number is adjusted and then FIX session is re-established. A new corrected resend request will be issued automatically.

For more info about catching the session level reject event see CustomCMEApplication::onSessionLevelRejectEvent implementation.

For more info about how to re-establish the FIX session with required sequence numbers see SampleApplication::midWeekReconnect method implementation.