QuickFIX to FIX Antenna™ C++ *Migration Guide*

Version 1.0.3.1 (2016-12-09)

www.b2bits.com B2BITS <epam>

Table of Contents

Overview

B2BITS provides FIXAntenna QF add-on package for FIX Antenna™ C++ library which helps to migrate from QuickFIX to FIX Antenna™ C++. The package includes C++ classes which implement QuickFIX interface. The migration to FIX Antenna™ engine essentially becomes a replacement of the library. The package also contains tools that help dealing with QuickFIX FIX protocol definition in XML.

FIXAntenna QF Adaptor supports import of customized dictionaries from QuickFIX. This is done by using an XML conversion tool which is part of the migration package (see Converting customized QuickFIX protocol dictionaries).

The user migrating to FIX Antenna™ can leverage Certified FIX Connections to major exchanges and ECNs still using QuickFIX business object model. This is achieved by using C++ code generator, which is part of the migration package, that can convert B2BITS provided FIX custom protocol definitions into the typed C++ business messages in QuickFIX style (see Generating C++ business objects for standard or customized FIX protocol versions).

Supported QuickFIX features

- Typed C++ message objects and MessageCracker interface
- Transient and persistent FIX session storage
- FIX Session management (schedules)
- 4.x and 5.x FIX protocol versions
- FIX protocol customizations

The following features are not part of the QF adaptor package. You may consider using the QuickFIX classes instead

- No database persistence
- No built-in HTTP remote administration. Use a separate feature rich GUI tool for remote administration: FIXICC
- The following configuration properties currently not supported: StartDay, EndDay (StartTime and EndTime are supported).
- A number of low-level QuickFIX C++ interfaces related to message store or logging details are not supported. The FIX Antenna™ engine library encapsulates its own optimized implementation of these interfaces and hides such low-level details from the user.

Performance Comparison

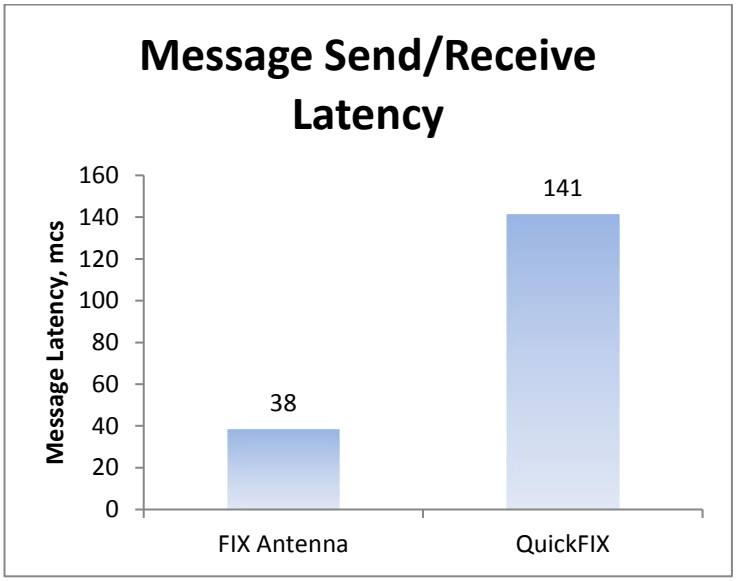
This section describes performance scenarios and numbers obtained using PerformanceTest application that is part of the migration package (please find the source code at: <path>\B2BITS\FIX Antenna C++\v2.5.0.1\QFAdaptor\examples\PerformanceTest). The application code is written using QuickFIX interface, the same source code is used when running the test with QuickFIX and FIX Antenna™ library.

FA 2.5 results

Latency Test

The test application establishes a FIX session on a local host using an instance of ThreadedSocketAcceptor/Initiator class. A FIX message (NewOrderSignle) is repeatedly sent within the session. The time of the FIX message travelling from the sending endpoint to the receiving endpoint is measured and reported. This is the time which is spent between Session::send() call and MessageCracker::onMessage().

FIX session uses persistent disk message store.



CPU: AMD Phenom II X4 3.0 GHz

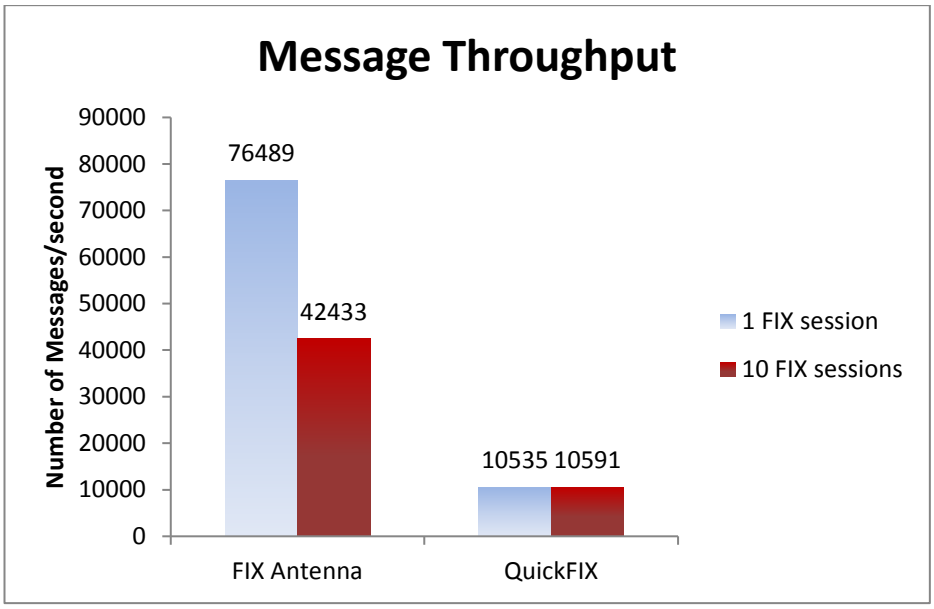
Average message latency with FIX Antenna™: 38 mcs (min: 34, max: 276)

Average message latency with QuickFIX: 146 mcs (min: 141, max: 14528)

Conclusion: The message delivery time with FIX Antenna™ QF Adaptor is 3.7 times less than that of the QuickFIX library.

Throughput Test

The test application establishes FIX sessions on a local host using an instance of SocketAcceptor/Initiator class. A FIX message (NewOrderSignle) is repeatedly sent into each of the created sessions. The test ends once all of the 1,000,000 messages are delivered to the receiving endpoint. The test calculates the throughput that is the number of messages delivered per 1 second.



Conclusion: The message throughput of FIX Antenna™ with QF adaptor is over 7 times more than that of the QuickFIX using the single instance of SocketAcceptor/Initiator class that hosts the sessions. With FIX Antenna™ you don't need to decide how many of the SocketAcceptor instances to launch in order to load your CPU cores with enough work. The scalability is achieved automatically regardless how the session grouping within particular acceptor/initiator instances.

FA 2.7 results

HP ProLiant DL 360, 2.8GHz, 2 CPU, 12 cores, x64, 16Gb RAM

Linux: Fedora 14 x64

B2BITS QF Adaptor results:

NullMessageStore, ThreadedSocketsInitiator/Acceptor, SocketNodelay=Y

```
---  Message latency  ---
Count: 100000
Min: 15 uSec
Max: 207 uSec
Avg: 20 uSec
```

Throughput

1 Session

NullMessageStore, SocketsInitiator/Acceptor, SocketNodelay=N

```
Messages sent: 1000000
Concurrent FIX sessions: 1
```

```
Throughput: 394802 messages/sec.
```

10 Sessions

```
Messages sent: 1000000

Concurrent FIX sessions: 10

Total Throughput: 375713 messages/sec.

Avg Throughput per session: 37571 messages/sec.
```

Migration Step-by-Step

Overview

Typically in order to migrate you need to do the following steps:

1. Install FIX Antenna™
2. Install FIX Antenna™ QF adaptor
3. Update your project to include QF adaptor source files and add a QF adaptor initialization call
4. Convert QuickFIX configuration files into FIX Antenna™ format
5. Compile and run

If you used custom FIX dictionaries:

6. Convert the dictionaries into FIXDIC format
7. An optional extra step: generate C++ business objects for custom FIX protocol

1. Migration Package Installation

Pre-requisites

In order to use the tools included with the package, ensure the following is installed:

1. Perl version 5.6 or later. Under Windows we recommend to use the ActivePerl distribution. In order to check that Perl is available, use the following command line:
   - perl –version
2. Java 1.5 or higher. In order to check that java is available, use the following command line:
   - java –version

   Note: for Linux the Sun Java or OpenJDK packages are recommended. The “gij (GNU libgcj) 1.4 ” was found to show significantly slower performance with the XML conversion tool.

Microsoft Visual Studio™ and GNU C++ compilers are supported. The version of the compiler should be the same as used by your FIX Antenna™ C++ package.

Installation

1. Install FIX Antenna™ C++ package. For information on how to get it please refer to the web site.
2. Unpack and place the “QFAdaptor” folder with FIX Antenna™ QF adaptor into the FIX Antenna™ installation at "<path>\B2BITS\FIX Antenna C++\v2.5.0.1". The version number may be different.

In the FIX Antenna™ QF Adaptor package all the C++ classes are enveloped in “QFA::” namespace. Additional define overrides “FIX” namespace definition and substitutes it with “QFA”.

To check that everything works OK, you may compile and run the example application found at "<path>\B2BITS\FIX Antenna C++\v2.5.0.1\QFAdaptor\examples\PerformanceTest\”. There is already an EXE binary in the “bin” folder built for Windows 32bit platform (using Microsoft Visual Studio® 2008), and you can run the performance test on Windows 32bit. For other platforms, please follow the instructions below:

Sample Project - Windows

1. Open PerformanceTest_vs2008.sln in Microsoft Visual Studio 2008

2. Build the solution
3. Go to PerformanceTest\bin\config and run config_convert_1session.cmd and config_convert_10session.cmd command files. These commands convert 1session.cfg and 10sessions.cfg configuration files from QuickFIX to FIXAntenna™ format. Output files are placed in the folder "converted"
4. Go to "bin" folder and run run_latency_test.bat, run_throughput_1session.bat, run_throughput_10sessions.bat.

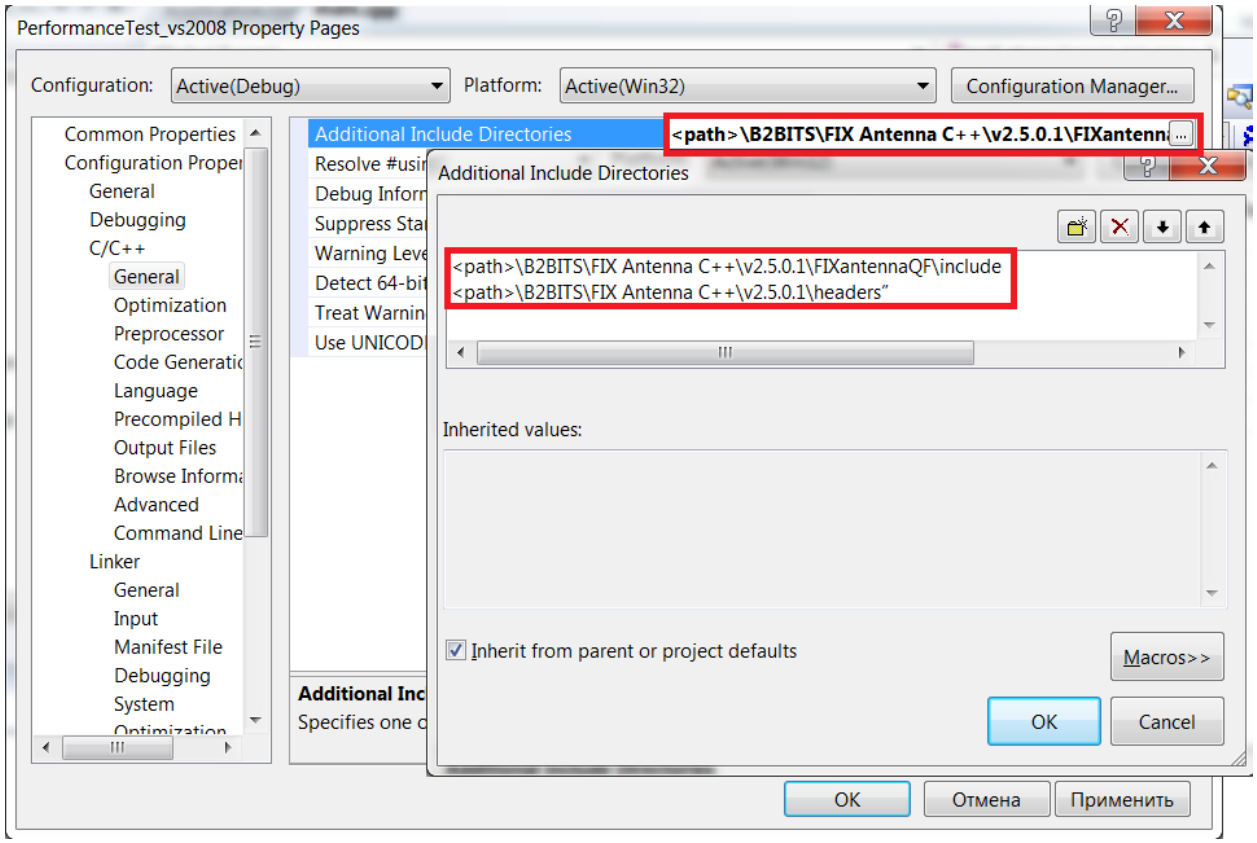
Sample Project - Linux

1. Go to PerformanceTest folder and run 'make'
2. Go to PerformanceTest\bin\config and run config_convert_1session.sh and config_convert_10session.sh command files. These commands convert 1session.cfg and 10sessions.cfg configuration files from QuickFIX to FIXAntenna™ format. Output files are placed in the folder "converted"
3. Go to "bin" folder and run run_latency_test.sh, run_throughput_1session.sh, run_throughput_10sessions.sh

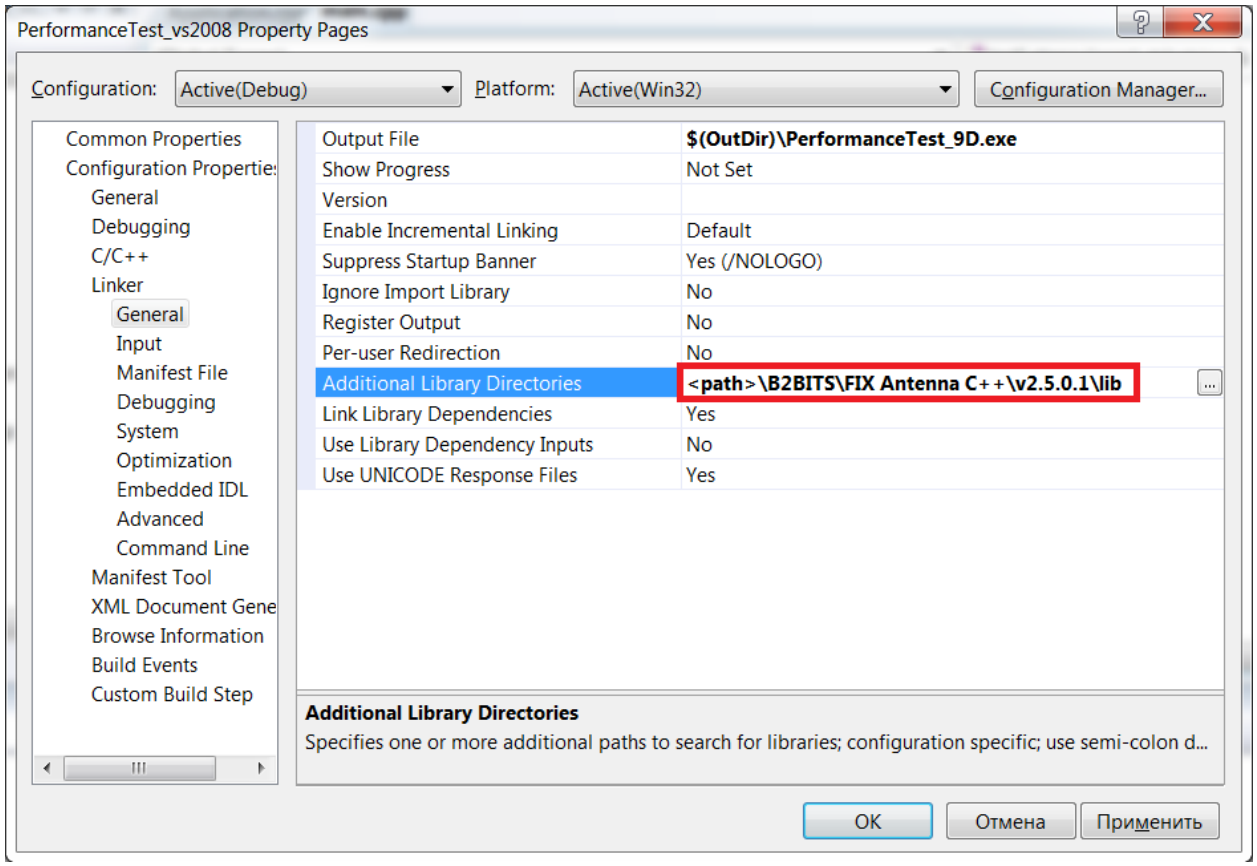
2. Updating Your Development Project

This chapter provides step-by-step instructions for changing a project that used QuickFIX C++ library and was created in Microsoft Visual Studio 2008. Similar steps will be necessary in case of other IDEs or platforms (for example, updating the project's makefile on Linux).

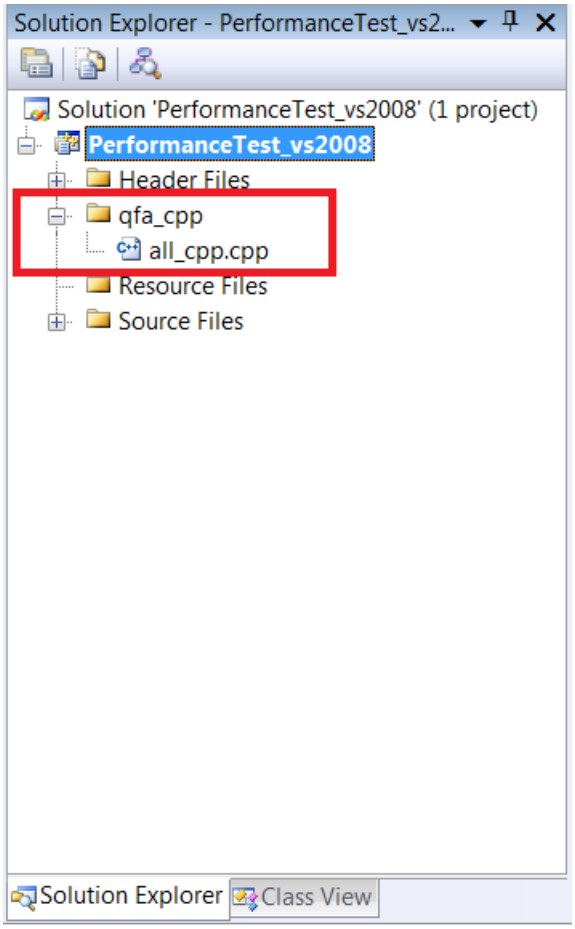
1. Follow the general FIX Antenna™ installation instructions
2. Remove the references to QuickFIX include and library folders from your project
3. Add the "<path>\B2BITS\FIX Antenna C++\v2.5.0.1\QFAdaptor\include" and "<path>\B2BITS\FIX Antenna C++\v2.5.0.1\headers" to your project's include paths



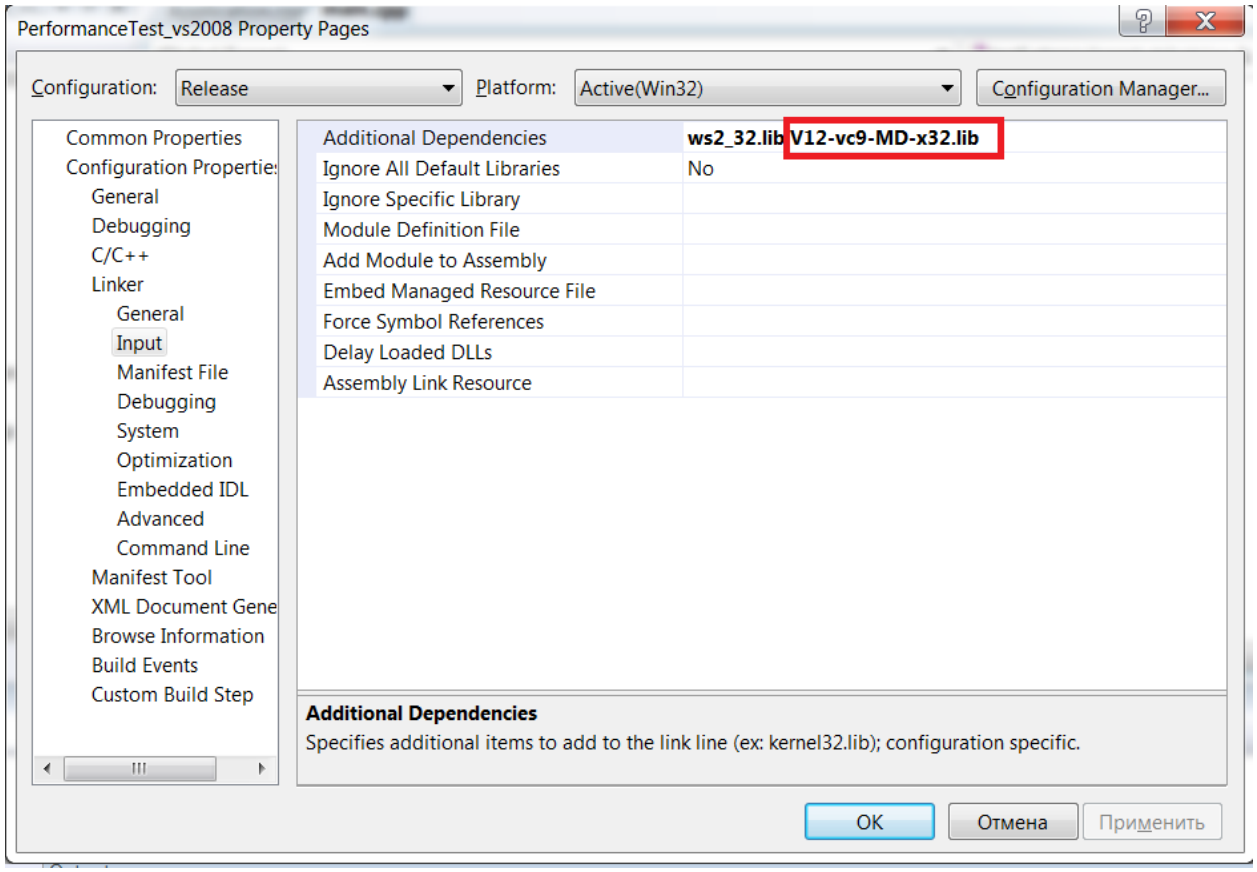
4. Add new library path to the project: "<path>\B2BITS\FIX Antenna C++\v2.5.0.1\lib”



5. Add the all_cpp.cpp file that is located in folder "<path>\B2BITS\FIX Antenna C++\v2.5.0.1\QFAdaptor\include\qfa\cpp\all_cpp.cpp” to your project so that they are compiled together with the rest of your project files.



6. Link the project with one of the FIX Antenna™ libs in the lib folder depending on release/debug configuration



You may refer to the sample project that is under "<path>\B2BITS\FIX Antenna C++\v2.5.0.1\QFAdaptor\examples\PerformanceTest\" to see how a project may be set up.

7. Insert a call to initialize QF adaptor and the instance of FIX Antenna™ library used by this adaptor before making further calls to it:

```
QFA::InitializeEngineAdaptor("<path>/qfa.adaptor.properties",
"<path>/engine.properties");
```

See the next chapter for the details on producing the .properties files.

8. In the end of the program, insert a call to shut down the adaptor and the instance of FIX Antenna™ library:

```
QFA::ShutdownEngineAdaptor();
```

3. Converting QuickFIX configuration files

The QuickFIX session configuration files are converted into B2BITS format using the "<path>\B2BITS\FIX Antenna C++\v2.5.0.1\QFAdaptor\tools\migrate_config\qfa_migrate_config.pl" script. In order to use this script, Perl interpreter version 5.6 or later has to be available. Under Windows we recommend to use the ActivePerl distribution.

In order to check the presence and version of Perl, use the following command line:

- perl --version

The migration tool consists of the following files:

- qfa_migrate_config.pl – Perl scripts
- engine.properties.template, qfa.adaptor.properties.template – Template files that are used to produce appropriate output files.

Tool invocation

To convert a QuickFIX .CFG file go to the folder where it resides and invoke the following command:
<path>\QFAdaptor\tools\migrate_config\qfa_migrate_config.pl <command like parameters>

Refer to the "example_conversion.cmd" file in the migrate_config folder to see how the command is used.

Below are the qfa_migrate_config.pl command line parameters:

```
Usage: qfa_migrate_config.pl <QuickFIX config file> <Engine RootDir setting> [Output
directory]

Mandatory parameters:

  "QuickFIX config file"            Source file to be converted

  "RootDir setting"                 Path to the fix engine root directory

                                    where the message store and log files will reside


Optional parameters:

  "Output directory"                Where to put the generated configuration files.

                                    If omitted, a new folder in current directory will be
created.
```

The script parses QuickFIX configuration file, maps the QuickFIX settings to the FIX Antenna™ settings and creates the output file. Notice the ERRORs or WARNINGs messages that may be generated by the conversion script when it fails to produce proper mapping and try to correct the cause.

The conversion script will create the following three files[1]:

- engine.properties – FIX Antenna™ engine global settings. The path to this file is then passed to the QFA::InitializeEngineAdaptor() call.

- qfa.adaptor.properties – The settings used specifically by the adaptor classes that provide QuickFIX interface for the user application. The path to this file is passed to the QFA::InitializeEngineAdaptor() call.

[1] The produced files can also be used by FIXEdge server as well without much change in the format. The leading "QFA." in the property names have to be replaced with appropriate FIXEdge prefix (refer to this FIXEdge guide).

- example.qfa.sessions[2] – This is the FIX session definition file. There can be multiple files of such kind used by an application. The path to the file is passed to an instance of FIX::SessionSettings class.

Note that two of the files, engine.properties and qfa.adaptor.properties, are used globally and are to be passed to the adaptor initialization call. There can be multiple *.qfa.sessions files used by your program and each conversion would produce the engine.properties and qfa.adaptor.properties files. If this is the case, review the produced engine.properties and qfa.adaptor.properties files (compare the file content) and chose the files for the QFA::InitializeEngineAdaptor() call.

IMPORTANT: The following properties of the global engine.properties file are updated by the migration script and are shared between all FIX connections:

- ListenPort – this is the setting copied from SocketAcceptorPort of the QuickFIX configuration file
- EngineRoot – the path used to resolved the log and message store relative paths
- Log.File.RootDir – the path to the additional log folder (set to be <EngineRoot>/logs)
- LogonTimeFrame, LogoutTimeFrame, AllowEmptyFieldValue, MessageMustBeValidated
- standard FIX XML files listed in DictionariesFilesList of engine.properties file

The following properties of the global qfa.adaptor.properties file are updated by the migration script and are shared between all FIX connections:

- Log.File.RootDir – the path to the folder where the QF adaptor stores its own logs (set to be <EngineRoot>/logs)

The following properties in qfa.adaptor.properties are to be set up by the user if necessary:

- new FIX dictionaries (besides standard ones in engine.properties) are listed in QFA.CustomVersion.<name> properties of qfa.adaptor.properties file

Backup FIX Connection

If the QuickFIX' configuration file specifies SocketConnectHost1/ SocketConnectPort1 pair, this is converted to the Backup connection properties of the FIX Antenna™ thus enabling to switch to this connection if the primary connection breaks.

Setting Session Properties on Run-time

This is an example of how to set up session properties programmatically:

```
void initExtraLowLatencyParams( FIX::Session& session )
{
    session.setSessionProperty( "TcpBufferDisabled", "true" );
    session.setSessionProperty( "StorageType", "null" );
    session.setSessionProperty( "SocketPriority", "AGGRESSIVE_SEND_AND_RECEIVE" );

    session.setSessionProperty( "AggressiveReceiveDelay", "0" );
    session.setSessionProperty( "ValidateCheckSum", "false" );
    session.setSessionProperty( "GenerateCheckSum", "false" );
}

void initSessionsLowLatency( const std::set<FIX::SessionID>& sessions )
```

[2] Note that Start/TerminateTimeUTC parameters are currently not supported by FIXEdge, use the parameters that specify a local time instead - Start/TerminateTime

```
{
    for( std::set<FIX::SessionID>::const_iterator it = sessions.begin();
         it != sessions.end();
         ++it )
    {
        FIX::Session* session = FIX::Session::lookupSession( *it );
        initExtraLowLatencyParams( *session );
    }
}
```

Main program:

```
{
……
    FIX::SessionSettings settings( file );
    Application application(true);
    FIX::NullStoreFactory storeFactory;
    FIX::ThreadedSocketAcceptor acceptor( application, storeFactory, settings );
    FIX::ThreadedSocketInitiator initiator( application, storeFactory, settings );

    initSessionsLowLatency( FIX::Session::getSessions() );
……
}
```

Log files to be watched on run-time

After starting the engine, the following log files will be created:

- <EngineRoot>/logs/qfa_engine_<digits>.log – the FIX Antenna™ engine log
- <EngineRoot>/logs/qfa_engine_adaptor.log – the QF adaptor log
- <EngineRoot>/logs/SENDER-TARGET_<digits>.conf – this file will contain the list of actual FIX Antenna™ session settings used when creating particular FIX session identified by <SENDER, TARGET>

4. Converting customized QuickFIX protocol dictionaries for use with FIX Antenna C++ engine

If your project uses customized FIX protocol definitions, you may convert them into FIXDIC format by using the utility "<path>\QFAdaptor\tools\xml_dict_conversion\dict_convert.cmd".

Replacing standard FIX version dictionaries

After conversion, you can replace the old dictionary with a new one in DictionariesFilesList the engine.properties.
Example: DictionariesFilesList =
../../../dict/b2bits/quickfix/fixdic40.xml;../../../dict/b2bits/quickfix/fixdic41.xml;**../../../dict/b2bits/quickfix/myfixdic42.xml**;../../../dict/b2bits/quickfix/fixdic43.xml;../../../dict/b2bits/quickfix/fixdic44.xml;../../../dict/b2bits/quickfix/fixdic50.xml;../../../dict/b2bits/quickfix/fixdic50sp1.xml;**../../../dict/b2bits/quickfix/myfixdic50sp2.xml;../../../dict/b2bits/quickfix/myfixdict11.xml**

Note that by replacing the standard FIXT11 dictionary all other FIX50 protocol versions are getting affected.

"FIX42 " and "FIX50SP2" are the FIX version identifiers that can be passed in string format to API functions that accept "const char* customFIXVersion" parameters, such as FIX::DataDictionary, FIX::Message, etc.

Besides that, the Engine::FIXVersion enumeration can be used to pass numeric constants like FIX42, FIX50SP2 to functions that accept Engine::FIXVersion parameter type.

Adding new custom FIX dictionaries

To add a custom dictionary, edit the XML file and change id attribute:

```
<fixdic xmlns="http://www.b2bits.com/FIXProtocol" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.b2bits.com/FIXProtocol fixdic.xsd"

        id="FIX42NSDQ" fixversion="4.2" title="FIX 4.2 (with errata 20010501)"
version="1.5.10" date="2011-11-04">
```

In case of FIX50 dictionary, also provide custom FIXT11 dictionary with a unique id.

Then edit the qfa.adaptor.properties file:

```
# For FIX4.x based protocols use: QFA.CustomVersion.<MyName> = <dictionary file>

# For FIX5.0 based protocols use: QFA.CustomVersion.<MyName> = <app dictionary
file>:<transport dictionary file>

#   Note that <transport dictionary file>'s should have unique <fixdic id=<value> inside.

# Examples:

QFA.CustomVersion.FIX42NSDQ = ../../../dict/fixdic/fixdic42_nsdq.xml

QFA.CustomVersion.FIX50SP2JSE =
../../../dict/fixdic/fixdic50sp2jse.xml:../../../dict/fixdic/fixdict11jse.xml
```

“FIX42NSDQ “ and “FIX50SP2JSE“ are the new FIX version identifiers that can be passed in string format to API functions that accept “const char* customFIXVersion” parameters, such as FIX::DataDictionary, FIX::Message, etc.

```
# XML additional fields file that provides protocol customization

QFA.CustomVersion.FIXcustom.AdditionalFieldsFileName = <path>/ customdict44_engine.xml
```

Creating Messages in C++ using customized FIX dictionaries

The following are examples of how to create messages using custom dictionaries. The syntax is sometimes different compared to the original QuickFIX.

Example 1. Creating a ExecutionReport message using the session’s FIX dictionary. Depending on what was configured for this session it might be a custom dictionary.

```
FIX::Session* session = FIX::Session::lookupSession(sessionID);
FIX::Message msg( *session, QFA::MsgType_ExecutionReport);
```

Example 2. Creating a ExecutionReport message using the FIXcustom dictionary configured in qfa.adaptor.properties file.

```
FIX::Message msg( “FIXcustomVersion”, QFA::MsgType_ExecutionReport);


FIX50::Message msg( “FIX50SP2JSE”, QFA::MsgType_ExecutionReport);
```

Example 3. Creating a ExecutionReport message using the FIXcustom dictionary configured in qfa.adaptor.properties file and C++ classes generated for this dictionary. See the following chapters for more info: Generate Message Classes bound to Custom FIX Protocols and New C++ namespace for custom FIX protocols

```
FIXcustom::Message msg( QFA::MsgType_ExecutionReport);

FIXcustom::ExecutionReport msg( … );

FIX42::ExecutionReport msg( … );
```

Example 4. Using DataDictionary class:

```
DataDictionary dataDictionary( "FIX43" );

DataDictionary dataDictionary( "..\FIX43.xml" );  // -----  FIX43 substring is identified in
the file name, and used to map to QFA FIX version identifiers. The file itself is ignored.

message.setString( str, true, &dataDictionary );
```

5. Converting customized QuickFIX protocol dictionaries for CPP code generator

Use the utility "<path>\QFAdaptor\tools\xml_dict_conversion\dict_convert_for_cpp.cmd" to convert QuickFIX XML definition into XML format suitable for further use with CPP code generator tool. This .CMD file is an example how to invoke the conversion tool.

6. Generating C++ business objects for standard or customized FIX protocol versions

Note: the CppGenerator.exe application is a Windows executable. When using the Linux version of the QF adaptor package, please unpack it on Windows and run the tool there.

Alternative 1. QuickFIX dictionary XML to CPP conversion

This method is the recommended way for generating CPP classes, it uses QuickFIX dictionaries as the source.

1. Go to <path>\QFAdaptor\tools\cpp_code_generator\quickfix-dict folder
2. Copy FIX40.xml, FIX41.xml, FIX42.xml, FIX43.xml, FIX44.xml, FIX50.xml, FIX50SP1.xml, FIX50SP2.xml, FIXT11.xml from quickfix/spec folder of your QuickFIX/C++ distribution to quickfix-input folder.
3. Add any custom dictionary files you would like to convert to the quickfix-input folder
4. Edit the quickfix-to-qfa-to-cpp.cmd file if any custom XMLs are used. Typically you would replace one of the existing FIXnn files with a custom one. Please refer to the following chapter if you would like to keep the base FIXnn version untouched and see your custom FIX protocol as a separate version of FIX protocol with new C++ message classes for business messages: Generate Message Classes bound to Custom FIX Protocols

When custom FIX protocol XML dictionaries are used, it may be desired to generate CPP message classes that will support the custom fields and groups defined in that dictionary. Then it will be possible to access these fields using get/set methods, for example, message.get(FIX42::MyNewField).

In contrast to standard FIX 4.x and 5.x protocols, the custom protocols are loaded from XML and registered in the FIX Antenna engine and QFA adaptor. The message classes need also to know how to find the custom dictionary when the message object is created.

The custom FIX protocols are defined in qfa.adaptor.properties file in QFA.CustomVersion… properties. Refer to the chapter for more info: 4. Converting customized QuickFIX protocol dictionaries for use with FIX Antenna C++ engine. The custom protocol is identified by an ID, which is a string, e.g. "CustomFIX44".

In order to pass this information to the CPP code generator, the "customFIXVer" attribute should be specified for the XML dictionary file that is passed to the tool in <path>\QFAdaptor\tools\cpp_code_generator\quickfix-to-qfa-to-cpp.cmd:

```
fixdic-input\fixdic44.xml@customFIXVer=MY_Custom_FIX44@param2=….
```

where MY_Custom_FIX44 is the name of the custom protocol defined in qfa.adaptor.properties file.

5. New C++ namespace for custom FIX protocol
6. Run quickfix-to-qfa-to-cpp.cmd command (on Linux use the .sh script)
7. Copy the contents of generated "qfa" folder to <path>\QFAdaptor\include\qfa. This will overwrite existing files in this folder.

Alternative 2. FIX Antenna XML to CPP conversion

This method is recommended in the case when one wants to generate the C++ classes for business messages directly out of the FIX Antenna FIXDIC XML dictionaries rather than out of the QuickFIX dictionaries.

The business message C++ classes can be generated from FIX Antenna ™ dictionaries using the <path>\QFAdaptor\tools\cpp_code_generator\qfa-generate.cmd utility (or .sh script on Linux). This command file can be further modified by the user who wants to produce the classes for customized FIX protocols. For example, consider changing the following command line in this .cmd file:

```
bin\CodeGenerator.exe fixdic-input\fixdic40.xml fixdic-input\fixdic41.xml fixdic-input\fixdic42.xml fixdic-input\fixdic43.xml fixdic-input\fixdic44.xml fixdic-input\fixdic50.xml fixdic-input\fixdic50sp1.xml fixdic-input\fixdic50sp2.xml fixdic-input\fixdict11.xml
```

and replace the text in bold (**`fixdic-input\fixdic44.xml`**) with a path to a custom FIX protocol definition. Such a definition can be produced from QuickFIX XML, refer to this chapter for the instructions: Converting customized QuickFIX protocol dictionaries for CPP code generator.

Note that the XMLs for all the FIX versions should be passed to this utility for processing at once (i.e. all files to be passed in the single command line, the code generator is not to be invoked for one individual file) because the code generator has to collect the definitions from all versions of the files to create consolidated lists in output files. This includes definitions for FIX tag numbers, message field values, etc.

After running the command, the "qfa" folder with the .H files is created. Copy the contents to the qfa folder located at: <path>\QFAdaptor\include\qfa

Generate Message Classes bound to Custom FIX Protocols

When custom FIX protocol XML dictionaries are used, it may be desired to generate CPP message classes that will support the custom fields and groups defined in that dictionary. Then it will be possible to access these fields using get/set methods, for example, message.get(FIX42::MyNewField).

In contrast to standard FIX 4.x and 5.x protocols, the custom protocols are loaded from XML and registered in the FIX Antenna engine and QFA adaptor. The message classes need also to know how to find the custom dictionary when the message object is created.

The custom FIX protocols are defined in qfa.adaptor.properties file in QFA.CustomVersion… properties. Refer to the chapter for more info: 4. Converting customized QuickFIX protocol dictionaries for use with FIX Antenna C++ engine. The custom protocol is identified by an ID, which is a string, e.g. "CustomFIX44".

In order to pass this information to the CPP code generator, the "customFIXVer" attribute should be specified for the XML dictionary file that is passed to the tool in <path>\QFAdaptor\tools\cpp_code_generator\quickfix-to-qfa-to-cpp.cmd:

```
fixdic-input\fixdic44.xml@customFIXVer=MY_Custom_FIX44@param2=….
```

where MY_Custom_FIX44 is the name of the custom protocol defined in qfa.adaptor.properties file.

New C++ namespace for custom FIX protocols

By default, the business message C++ classes are created in FIX**nn** namespace which corresponds to the FIX base protocol version taken from the XML. If you'd like to place your customized protocol classes into a separate distinct namespace, pass the XML file to bin\CodeGenerator.exe command as follows: `fixdic-input\fixdic44.xml@namespace=MY_Custom_FIX44@param2=….`

As long as the MessageCracker class cannot distinguish between the base FIX version and your custom version since the BeginString field of the message will contain the base FIX version number, you would have to do the message class conversion on your own, for example, consider modifying <path>\QFAdaptor\include\qfa\MessageCracker.h like follows:

```
#include <qfa/my_custom_fix44/MessageCracker.h>  // add this include

class MessageCracker :
      public FIX40::MessageCracker,
      public FIX41::MessageCracker,
      public FIX42::MessageCracker,
      public FIX43::MessageCracker,
      public FIX44::MessageCracker,
      public FIX50::MessageCracker,
      public FIX50SP1::MessageCracker,
      public FIX50SP2::MessageCracker,
      public FIXT11::MessageCracker,
     public MY_Custom_FIX44::MessageCracker       // add new class inheritance
{
public:
    virtual void crack( const Message& message, const SessionID& sessID)
    {
        Engine::FIXVersion ver = message.getMsgVer();
        switch(ver)
        {
            case Engine::FIX40:
                return static_cast<FIX40::MessageCracker*>(this)->
                    crack( static_cast<const FIX40::Message&>(message), sessID );
            case Engine::FIX41:
                return static_cast<FIX41::MessageCracker*>(this)->
                    crack( static_cast<const FIX41::Message&>(message), sessID );
            case Engine::FIX42:
                return static_cast<FIX42::MessageCracker*>(this)->
                    crack( static_cast<const FIX42::Message&>(message), sessID );
            case Engine::FIX43:
                return static_cast<FIX43::MessageCracker*>(this)->
                    crack( static_cast<const FIX43::Message&>(message), sessID );
            case Engine::FIX44:
              if( sessID.sessionQualifier_ == <My_custom_FIX44_initiator>"  ||
                  sessID.senderCompID_ == "<Specific sender>" ) // or put any other appropriate
condition here...
                  return static_cast< MY_Custom_FIX44::MessageCracker*>(this)->
                      crack( static_cast<const MY_Custom_FIX44::Message&>(message),
sessID );
             else
                return static_cast<FIX44::MessageCracker*>(this)->
                    crack( static_cast<const FIX44::Message&>(message), sessID );
            case Engine::FIX50:
                return static_cast<FIX50::MessageCracker*>(this)->
```

```
                crack( static_cast<const FIX50::Message&>(message), sessID );
            case Engine::FIX50SP1:
                return static_cast<FIX50SP1::MessageCracker*>(this)->
                    crack( static_cast<const FIX50SP1::Message&>(message), sessID );
            case Engine::FIX50SP2:
                return static_cast<FIX50SP2::MessageCracker*>(this)->
                    crack( static_cast<const FIX50SP2::Message&>(message), sessID );
            case Engine::FIXT11:
                return static_cast<FIXT11::MessageCracker*>(this)->
                    crack( static_cast<const FIXT11::Message&>(message), sessID );
        }
    }
```

7. Converting existing B2BITS FIX protocol customizations into QuickFIX C++ business objects

B2BITS can provide you with FIX protocol customizations used to connect to various destinations. Let's consider MICEX exchange as an example. There will be the micex_fix.xml file which defines additional fields used in some FIX messages. This file is to be mentioned in QFA.CustomVersions property of qfa.adaptor.properties file.

In order to obtain the C++ classes for customized business messages follow these steps:

1. Merge the micex_fix.xml customization with the base protocol, for example FIX44. Use the <path>\QFAdaptor\tools\xml_dict_conversion\dict_merge.cmd command line to do the merging.
2. Take the merged XML file and follow the instructions specified in chapter Generating C++ business objects for standard or customized FIX protocol versions

APPENDIX A. Configuration Reference

engine.properties File

Also refer to the latest version of documentation on the B2BITS web site: engine.properties or documentation in your FIX Antenna™ distribution: <path>\B2Bits\FIX Antenna C++\v2.3.12.17\doc\index.html

qfa.adaptor.properties File[3]

Property	Default Value	Description
QFA.CustomVersion.<Custom FIX ID>.<Name> = <App dict XML>:<Transport Dict XML>		Denotes a custom FIX protocol
Log.File.RootDir		
Log.Device	File Console	Target devices
Log.DebugIsOn	false	Turns on/off logging on the debug level
Log.NoteIsOn	true	Turns on/off logging on the notice level
Log.WarnIsOn	true	Turns on/off logging on the warning level
Log.ErrorIsOn	true	Turns on/off logging on the error level
Log.FatalIsOn	true	Turns on/off logging on the fatal level
Log.Cycling	false	Turns cycling on/off.
Log.Cycling.Ignore	3	Number of repeating records to be placed to log before cycling is started.
Log.Cycling.BlockSize	10	Number of repeating records to be accumulated (hidden) before writing the "cycle record" to the log.
Log.Cycling.Multiplier	10	Multiplier for the Block Size. If BlockSize number of messages is accumulated and the same message still appears then next BlockSize is calculated as the previous one multiplied by Multiplier.
Log.File.Name		File name. If more than one category uses files with the same name the same file will be used simultaneously.
Log.File.Recreate	false	If true then file will be recreated on each start. If false then new records will be appended to the

[3] The corresponding FIXEdge parameters are described in this guide

		current file.
Log.File.AutoFlush	true	If set to true then the buffer is flushed after each logging call. If set to false then flush is not called. Setting to true decreases program performance; setting to false increases a risk of record loss in the event of program failure.

qfa.sessions File[4]

Property	Default Value	Description
QFA.Sessions		This parameter determines the names of FIX session-acceptors. For each such a session it is necessary to define the parameters described below. The format is 'session.X.ParameterName' where 'X' is the name of session.
QFA.Session.<Name>.Version		FIX protocol version. Allowed values are FIX40, FIX41, FIX42, FIX43, FIX44, FIX50, FIX50SP1, FIX50SP2 or custom protocol names as specified in qfa.adaptor.properties file
QFA.Session.<Name>.Role		Initiator/Acceptor
QFA.Session.<Name>.Username		Username for FIX Session authentication
QFA.Session.<Name>.Password		Password for FIX Session authentication
QFA.Session.<Name>.SenderCompID		SenderCompID (Assigned value is used to identify a firm sending message).
QFA.Session.<Name>.TargetCompID		TargetCompID (Assigned value is used to identify a receiving firm).
QFA.Session.<Name>.Host		Network address of the computer, to which connection is established.
QFA.Session.<Name>.Port		Port's network number on the computer, to which connection is established.
QFA.Session.<Name>.HBI	60	Time interval (in seconds) between Heartbeat messages. The recommended value is 10 seconds for dedicated connections or private networks. Trading connections via the internet will require calibration. '0' means that no Heartbeat messages will be sent.
QFA.Session.<Name>.InSeqNum	0	The initial incoming sequence number. The first incoming message is expected to have the specified sequence number. The default value will be used when set to zero.
QFA.Session.<Name>.IntradayLogoutTolerance	false	An option not to reset sequence numbers after a Logout. The party sending a Logout should initiate session recovery by sending a Logon message with SeqNum = + 1; expecting reply Logon withSeqNum = + 1. If a gap is detected, standard message recovery or gap filling processes arise. This property overrides the 'IntradayLogoutTolerance' property in

[4] Note that this configuration file uses a syntax which is close to that of the FIXEdge product. This is helpful when someone wants to migrate from QuickFIX to FIXEdge. The corresponding FIXEdge parameters are described in "FIX Edge - AdminGuide.html" available in the FIXEdge product installation.

		'engine.properties' for this session. **Note** : The default value is taken from engine.properties
QFA.Session.<Name>.OutSeqNum	0	The initial outgoing sequence number. The first outgoing message will be sent with the specified sequence number. If '0' then the default value is used.
QFA.Session.<Name>RejectMessageWhileNoConnection	false	When true, application messages will be rejected, when session unable to send them during specified period or after being disconnected.
QFA.Session.<Name>.Description		Session's description.
QFA.Session.<Name>.StartTime		Local time to start the session (HH:MM). If the start-up time is greater than the specified value then the session will be started immediately. Note: this property is optional.
QFA.Session.<Name>.TerminateTime		Local time to terminate this session (HH:MM). If the start-up time is greater than the specified value then the value will take effect. Note: this property is optional.
QFA.Session.<Name>.StartTimeUTC		Start time in UTC
QFA.Session.<Name>.TerminateTimeUTC		Terminate time in UTC
QFA.Session.<Name>.SourceIPaddress	localhost	The expected value of the source IP address. If the real value is not equal to the expected one then the session is disconnected without sending a message and an error condition is generated in the log output.
QFA.Session.<Name>.RecreateOnLogon	Initiator: true Acceptor: false	If set to true and an attempt to connect fails, QFA adaptor will continue to ask the FIX Engine to establish the connection until it is successfully done.
QFA.Session.<Name>.RecreateOnLogout	Initiator: false Acceptor: true	If set to false then session is removed from the list of sessions after successful disconnection. If set to true then it will be recreated after disconnection. **Note** : recreation will take place only if disconnection is initialized by the counter-party.
QFA.Session.<Name>.ReconnectInterval	30	Time interval at which the ReconnectOnLogon/Logout attempts will be made for session initiators. For acceptors it is always set to 0 meaning to repeat immediately. So for example an acceptor with RecreateOnLogout=true is reacreated and is ready for a new connection right after the logout.
QFA.Session.<Name>.ForceReconnect	true	Extends the session connection retry mechanism in the FIX Engine to the logon attempts. The engine will try to establish a connection if the remote endpoint is not

		available. Without this settings, the engine reconnects only when an already established session breaks. The number of retry attempts is specified in the Reconnect.MaxTries setting of the engine.property file. The time interval is set by the Reconnect.Interval property in the same file. With this setting enabled, the connection may switch to backup connection at the logon time, if the primary endpoint is not available.
QFA.Session.<Name>. SenderLocationID		FIX tag 142 - assigned value used to identify specific message originator's location (i.e. geographic location and/or desk, trader).
QFA.Session.<Name>. TargetLocationID		FIX tag 143
QFA.Session.<Name>. ForceSeqNumReset	0	This parameter allow to automatically resolve sequence gap problem. When mode is ON session uses 141(ResetSeqNumFlag) tag in sending/confirming Logon message to reset SeqNum at the initiator or acceptor. "0" - Disable ForceSeqNumReset mode "1" - Enable SeqNum reset at first time of session initiation "2" - Enable SeqNum reset for every session initiation
QFA.Session.<Name>. ResetSeqNumAtScheduledStartTime	false	Perform FIX session sequence numbers at the scheduled session's start time (do so when the session happens to be still alive at the scheduled time)
QFA.Session.<Name>. IntradayLogoutTolerance	true	If true, don't reset sequence numbers on logout
QFA.Session.<Name>. KeepConnectionState	false	When true, primary to backup (and back) connection switching continue using existing message storage
QFA.Session.<Name>. SocketPriority	EVEN	EVEN – use a pool of worker threads for network I/O AGGRESSIVE_SEND_AND_RECEIVE – use dedicated thread for network I/O to achieve a minimal latency. It is set to true by the ThreadedSocketAcceptor/Initiator automatically, but may be overridden in the configuration file
StorageType	persistent	Type of the message storage to use. persistent – disk storage persistentMM – memory mapped file transient – in-memory storage null – no storage Appropriate value is chosen by the FileStore/MemoryStoreFactory, but may be overridden in the configuration file.

QFA.Session.<Name>. EncryptMethod		1 - NONE 2 - DES 3 - PKCS_DES 4 - PGP_DES 5 - PGP_DES_MD5 6 - PEM_DES_MD
QFA.Session.<Name>. TcpBufferDisabled	true	If true, the Nagle algorithm on TCP/IP connection is disabled to achieve the minimal latency of message sending. It is set to true by the ThreadedSocketAcceptor/Initiator automatically, but may be overridden in the configuration file
QFA.Session.<Name>. IgnoreSeqNumTooLowAtLogon		This parameter allows to resolve 'seqNum too low' problem at logon. When true - session continues with the received seqNum.
QFA.Session.<Name>. Backup.Host		Host name to be used for a backup FIX connection (valid for session initiators)
QFA.Session.<Name>. Backup.Port		Port number to be used for backup FIX connection (valid for session initiators)
QFA.Session.<Name>. Backup.HBI		Backup session setting.
QFA.Session.<Name>. Backup.SenderSubID		Backup session setting.
QFA.Session.<Name>. Backup.TargetSubID		Backup session setting.
QFA.Session.<Name>. Backup.SenderLocationID		Backup session setting.
QFA.Session.<Name>. Backup.TargetLocationID		Backup session setting.
QFA.Session.<Name>. Backup.IntradayLogoutTolerance		Backup session setting.
QFA.Session.<Name>. Backup.ForceSeqNumReset		Backup session setting.
QFA.Session.<Name>. Backup.ForceReconnect		Backup session setting.
QFA.Session.<Name>. Backup.IgnoreSeqNumTooLowAtLogon		Backup session setting.
QFA.Session.<Name>. Backup.EnableAutoSwitchToBackupConnection		When true, automatic switch mode is enabled. By default automatic switch mode is disabled.
QFA.Session.<Name>. Backup.EnableCyclicSwitchBackupConnection		When true, connection will be switched from primary to backup and back until connection will be established.
QFA.Session.<Name>. ActiveConnection	primary	backup - The session connects via backup connection restore - The session connects to the previous active connection Otherwise, the session connects via primary connection.
QFA.Session.<Name>.CustomLogonFileName		Path to the file containing FIX Logon message that will be sent to the counterparty when initiating the connection. The message is in binary

	format. If a relative path is specified, the root folder is taken from the engine.properties EngineRoot setting

Contact us

sales@btobits.com

Phone: +1-888-378-0666

Global Headquarters
US Client Support and Delivery Center

EPAM Systems, Inc
41 University Drive
Suite 202
Newtown, PA 18940
Phone: +1-267-759-9000
Fax: +1-267-759-8989