# FIXEdge

*Business Layer Plug-in Interface*

# Table of Contents

## Overview

FIXEdge business layer provides open interface for plug-ins called ‘DLL handlers’ or simply ‘handlers’. To deploy custom functionality into the FIXEdge on business layer user must do the following:

- Develop a dynamic library (DLL) and implement FixServer::BusinessLayer::HL::Handler interface.
- Put library to the directory accessible by FIXEdge
- Add handler description to the business layer configuration file
- Create correspondent rule with action that fires plug-in method

# Plugin-in interface

### *Handler interface*

Handler interface is defined in ‘B2BITS_Handler.h’ and ‘B2BITS_HandlerDll.h’. The main handler interface is Handler:

```
/// Handler interface
struct Handler
{
   /// Initialize Handler. This methos is called once for handler each time business layer
   /// is initialized. Thi smethod is NOT called whe rule is applied to the message.
   /// @param fixEdgeProps set of the FixEdge properties.
   /// @param handlerProps set of the Handler's properties.
   /// @param pHelper handler helper
   virtual void init( const StringProperties &fixEdgeProps, const StringProperties &handlerProps,
HandlerHelper *pHelper ) = 0;

   /// Registers message sender. Registered pointer is valid until detach() or destroy()
   /// is called.
   /// @param pObserver - message sender object.
   virtual void attach( MessageSender *pObserver ) = 0;

   /// De-registers message sender. Do not use pointer after this method is called.
   virtual void detach() = 0;

   /// Destroys handler. Note detach() has already called.
   virtual void destroy() = 0;

   /// Processes message. Method is called each time rule is successully applied to
   /// the message if correspondent handler call exists in rule's action
   /// @param msg Message being processed
   /// @param pIdentifier contains session ID if message was received from transport
   /// adaptor. Otherwise (i.e. message is received from FIX session) equal to NULL.
   /// @note If message is received from FIX session then SenderCompID and TargetCompID can
   /// be extracted from message.
   /// @param isSourceID If parameter has true value then pIdentifier is the source identifier.
   /// @param props contains additional parameters.
   /// @return false if handler unable to process this message
   virtual bool process( const Engine::FIXMessage &msg, const char *pID, bool isSourceID, const
StringProperties *props = NULL ) = 0;

   /// Destructor. Desrtoys hander.
   virtual ~Handler() {}
};
```

It is required to implement this interface.

### *Processing message*

Any time rule is executed and action is fired the Handler::process method is called. Message, which is processed, is passed to that method as an input parameter. Message can be received from FIX Session or other transport (i.e. from client). In the first case FIX Session ID can be extracted from message itself (SenderCompID and TargetCompID). In the second case the client ID is passed as an input parameter to the method. If message is received from FIX Session then input parameter is NULL.

### *Sending message*

The MessageSender is to be used for message sending.

```
/// Message sender interface. Observer.
struct MessageSender
{
   /// Enqueues message to be send to the client session i.e. via transport adapter
   /// @param msg Message to be sent
   /// @param pIdentifier can contains the transport adapter client ID where message will be sent
   /// or source identifier of FixSession or TA client
   /// @ bIsClientID If parameter has false value then pIdentifier is the source identifier
   virtual void enqueue( const Engine::FIXMessage &msg , const char *pIdentifier, bool bIsClientID = true) =
0;

   /// Enqueues message to be send to the FIX session.
   /// @param msg Message to be sent
   /// @param pSenderCompID target FIX session's SenderCompID
   /// @param pTargetCompID target FIX session's TargetCompID
   /// @note do not use this message to send message to the transport adapter session
   virtual void enqueue( const Engine::FIXMessage &msg, const char *pSenderCompID, const char
*pTargetCompID ) = 0;

   /// Verifies that any message was enqueued for passed target
   /// @param pIdentifier can contains the transport adapter client ID or source identifier
   /// @ bIsClientID If parameter has false value then pIdentifier is the source identifier
   /// @return true if message is saved to queue, false – otherwise
   virtual bool isMsgEnqueued(const char *pIdentifier, bool bIsClientID = true) = 0;

   /// Verifies that any message was enqueued for passed target
   /// @param pSenderCompID FIX session's SenderCompID
   /// @param pTargetCompID FIX session's TargetCompID
   /// @return true if message is saved to queue, false - otherwise
   virtual bool isMsgEnqueued( const char *pSenderCompID, const char *pTargetCompID ) = 0;
};
```

Message is not sent immediately after MessageSender::enqueue is sent. It is scheduled for sending instead and sent right after all business layer rules are executed.
Message can be sent to FIX Session or Client. Depending on desired destination different instance of enqueue methods is to be used.
There is no restriction for the number of messages to enqueue.

## Deployment

It is enough to put DLL to the FIXEdge ‘bin’ directory; in that case it can be addressed in definition section without path. However it is possible to put library anywhere on the computer; in this case the full path to the file is to be specified in DllName e.g. DllName=”c:/MyPluginDir/MyPlugin.dll”.

# Business Layer rules

### *Define Plug-in*

```
<DllHandlers>
        <Handler
                Name="TestHandler"
                Description="Test Handler description"
                DllName="./SimplePlugin_71.dll"
                VerifyHandlersVersion="true" />
</DllHandlers>
```

### *Call Action*

```
<Action>
        <HandlerAction Name="TestHandler" />
</Action>
```

## Sample plug-in

```
/*
 * <COPYRIGHT>
 * $Revision: 1.57 $
 * (c) B2BITS 2009. B2BITS is an abbreviation of
* Business to Business Information Technology Services corporation.
* "Licensor" shall mean B2BITS.
*
* This software is for the use of the paying client of B2BITS (which may be
* a corporation, business area, business unit or single use subject to
* licence terms) to whom it was delivered (the "Licensee")and no other party,
* and any use beyond this business area is contrary to the terms of the licence grant.
*
* The Licensee acknowledges and agrees that the Software and Documentation
* (the "Confidential Information") is confidential and proprietary to
* the Licensor and the Licensee hereby agrees to use the Confidential
* Information only as permitted by the full licence agreement between
* the two parties, to maintain the confidentiality of the Confidential
* Information and not to disclose the confidential information, or any part
* thereof, to any other person, firm or corporation. The Licensee
* acknowledges that disclosure of the Confidential Information may give rise
* to an irreparable injury to the Licensor in-adequately compensable in
* damages. Accordingly the Licensor may seek (without the posting of any
* bond or other security) injunctive relief against the breach of the forgoing
* undertaking of confidentiality and non-disclosure, in addition to any other
* legal remedies which may be available, and the licensee consents to the
* obtaining of such injunctive relief. All of the undertakings and
* obligations relating to confidentiality and non-disclosure, whether
* contained in this section or elsewhere in this agreement, shall survive
* the termination or expiration of this agreement for a period of five (5)
* years.
*
* The Licensor agrees that any information or data received from the Licensee
* in connection with the performance of the support agreement relating to this
* software shall be confidential, will be used only in connection with the
* performance of the Licensor's obligations hereunder, and will not be
* disclosed to third parties, including contractors, without the Licensor's
* express permission in writing.
*
* Information regarding the software may be provided to the Licensee's outside
* auditors and attorneys only to the extent required by their respective
* functions.
*
 * </COPYRIGHT>
 */

/// Defines the entry point for the DLL application.

#include "B2BITS_Exception.h"
#include "B2BITS_HandlerDll.h"
#include "B2BITS_Handler.h"

#include <assert.h>

#if defined(_WIN32)
```

```
 #include <windows.h>
 #define DLLEXPORT __declspec(dllexport)
#else // LINUX
 #define DLLEXPORT
#endif


using namespace std;
using namespace FixServer::BusinessLayer::HL;

namespace {

static Handler* gHandler(NULL);
static bool gIsInitialised(false);
static const size_t VERSION_LENGTH = 32;
static const char VERSION[VERSION_LENGTH] = "2.0";

class SimplePlugin : public Handler
{
public:

    /// Initialize Plugin
    virtual void init( const StringProperties &fixEdgeProps, const StringProperties &handlerProps,
HandlerHelper *pHelper );

    /// Attach message sender
    virtual void attach( MessageSender *pObserver );

    /// Detach message sender
    virtual void detach();

    /// Destroy plugin
    virtual void destroy();

    /// Process message
    virtual void process( const Engine::FIXMessage &msg, const char *pID, bool isSourceID, const
StringProperties *props = NULL);

    /// Constructor
    SimplePlugin();

    /// Destructor
    virtual ~SimplePlugin();

private:

    // Logger
    HandlerLogger *pLogger_;

    // Message sender interface
    MessageSender *pObserver_;

    // Session ID
    std::string sender_;
    std::string target_;
};
```

```
//////////////////////////////////////////////////////////////////////////

#define LOG_NOTE(x) pLogger_->note((x), strlen((x)));

void SimplePlugin::init( const StringProperties &fixEdgeProps, const StringProperties &handlerProps,
HandlerHelper *pHelper )
{
   if( gIsInitialised )
   { return; }
   // create log category
   pLogger_ = pHelper->createLogCategory("SimplePlugin");

   // Set some dummy values
   sender_ = "snd";
   target_ = "tgt";

   gIsInitialised = true;
   LOG_NOTE("Plugin has been initialized.");
}

void SimplePlugin::attach( MessageSender *pObserver )
{
   // Store sender internally
   pObserver_ = pObserver;
   LOG_NOTE("Message sender registered to plugin");
}

void SimplePlugin::detach()
{
   // After detach is called sender should not be used
   pObserver_ = NULL;
   LOG_NOTE("Message sender unregistered from plugin");
}

void SimplePlugin::destroy()
{
   if( !gIsInitialised )
   { return; }

   //
   // Collect garbage here.
   //

   gIsInitialised = false;
   LOG_NOTE("Plugin destroyed");
}

void SimplePlugin::process( const Engine::FIXMessage &msg, const char *pID, bool isSourceID, const
StringProperties *props)
{
   // We cannot send message if sender is not attached
   if( NULL == pObserver_ )
   { return; }

   // send message to certain session
```

```
    pObserver_->enqueue( msg, sender_.c_str(), target_.c_str() );
    LOG_NOTE("Message scheduled for sending to FIX session <'snd', 'tgt'>");
}

SimplePlugin::SimplePlugin()
: pLogger_(NULL), pObserver_(NULL)
{}

SimplePlugin::~SimplePlugin()
{}

}

////////////////////////////////////////////////////////////////////////

/// Version method
extern "C" DLLEXPORT size_t getVersion( char *buffer, size_t bufferLen )
{
    size_t verLen = min( bufferLen, strlen(VERSION) );
    strncpy(buffer, VERSION, verLen);
    buffer[verLen] = 0;
    return verLen;
}

/// The Factory method.
extern "C" DLLEXPORT FixServer::BusinessLayer::HL::Handler* getInstance()
{
    if( NULL == gHandler )
    { gHandler = new SimplePlugin(); }
    assert( NULL != gHandler );
    return gHandler;
}

/// Release method.
extern "C" DLLEXPORT void release()
{
    delete gHandler;
    gHandler = NULL;
}

////////////////////////////////////////////////////////////////////////

#if defined (_WIN32)
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID /*lpReserved*/
                       )
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
      break;
    case DLL_THREAD_ATTACH:
      break;
    case DLL_THREAD_DETACH:
      break;
```

```
    case DLL_PROCESS_DETACH:
      release();
      break;
    }
    return TRUE;
}
#else // _LINUX

/* library's initialization function. */
void _init()
{}

/* library's cleanup function */
void _fini()
{
    release();
}

#endif
```

## Sample Business Layer properties file

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
	FIXEdge - the XML Configuration file

	$Revision: 1.14 $
-->
<!DOCTYPE FIXEdge SYSTEM "BusinessLayer.dtd">
<FIXEdge>
	<BusinessLayer>
		<DllHandlers>
			<Handler Name="Handler1"
					Description="Simple Handler description"
					DllName="./SimplePlugin_71.dll"
					VerifyHandlersVersion="true" />
		</DllHandlers>
		<!-- Handle message from all FIX session by Handler1 -->
		<Rule>
			<Source>
				<FixSession SenderCompID=".*" TargetCompID=".*" />
			</Source>
			<Condition>
				<EqualField Field="35" Value="D" />
			</Condition>
			<Action>
				<HandlerAction Name="Handler1" />
			</Action>
		</Rule>
	</BusinessLayer>
</FIXEdge>
```

## 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