FIXEdge BL Handlers Developer Guide

Overview

FIXEdge business layer provides open interface for plug-ins called ‘DLL handlers’ or simply ‘handlers’. To deploy custom functionality into the FIX Edge 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 FIX Edge
- Add handler description to the business layer configuration file
- Create correspondent rule with action that fires plug-in method

Handler interface

Handler interface is defined in ‘B2BITS_Handler.h’ and ‘B2BITS_HandlerDll.h’. Any developed handler should implement Handler interface:

```
struct Handler
{
    /// Meyer's case-insensitive comparision functor.
    struct CStrCaseInsComp : public std::binary_function< const char*, const char*, bool >
    {
        bool operator()( const char* lhs, const char* rhs ) const
        { return (0 < strcmp(lhs, rhs)); }
        };
    /// map of pairs <key, value> for properties
        typedef std::map< const char*, const char*, CStrCaseInsComp > StringProperties;
    /// 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.
    /// @note pObserver not enqueues messages from handler's internal threads (from handler caller threads only)
    virtual void attach( MessageSender *pObserver ) = 0;
    /// De-registers message sender. Do not use pointer after this method is called.
    virtual void detach() = 0;
    /// Registers message sender. Registered pointer is valid until detachAsyncSender() or destroy()
    /// is called.
    /// @param pObserver - message sender object.
    virtual void attachSyncSender( SyncMessageSender *pObserver ) = 0;
    /// De-registers message sender. Do not use pointer after this method is called.
    virtual void detachSyncSender() = 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) contains FIX session name from config or
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 execResult can be used
    ///              for replacing original message (execResult.resultMsg_ - ownership is transmitted) for
further processing or
    ///              for making custom reject with (execResult.description_) if message processing should be
terminated
    /// @return false if message processing should be terminated in this rule
    virtual bool process( const Engine::FIXMessage &msg, const char *pID, bool isSourceID, const
StringProperties *props = NULL, ExtendedExecutionResult* execResult = NULL ) = 0;
    /// Destructor. Desrtoys hander.
    virtual ~Handler() {}
        static const char* findProperty(const StringProperties& props, const char* value)
        {
                Handler::StringProperties::const_iterator iter = props.find(value);
                if (iter != props.end())
                {
                        return (*iter).second;
                }
                return "";
        }
```

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 and Configuration

It is enough to put DLL to the FIX Edge ‘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”.

Register a plug-in in BL_Configuration.xml:

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

Configure some business rule to execute a Handler:

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

For more info about business layer configuration please see Configuring Business Layer

Sample

Sample handler code is available at src_Handler folder of SDK