Developer Guide

External Price Feed SDK Guide

Version 2.1, 13th February 2013

Author
John Livingstone, Lukas Luthy

Distribution

Revision History

Version	Change description	Made by
Draft 0.1, 29th Sept 2011	First draft of document	John Livingstone
Version 2, 12th Dec 2012	Added strategy handling and market status	Lukas Lüthy
Version 2.1, 13th Feb 2013	Addition of InstrumentExchangeId	Marcin Ziolek

Table of Contents

Introduction

Purpose

A guide for developers using the Actant External Price Feed SDK to develop their own price feeds.

It covers the installation of the SDK, building the sample price feed application and the installation of the External Price Feed IIO to run the sample code.

Audience

Aimed at developers using the SDK. It describes the process of writing a Price Feed at the code level.

Glossary of Terms

Term	**Definition**
IIO	Interface Implementation Object. External object that loaded by Actant Quote at startup to handle Exchange Connectivity
ClientModule	A DLL module loaded by the External Price Feed IIO. It will be written externally to Actant and be capable of delivering prices to Actant

Overview

Overall Architecture

The diagram above shows the role the External Price Feed IIO plays within Actant Quote. It co-exists with other IIOs which connect to markets, download instrument details and handle the sending of quotes and orders to the market.

The External Price Feed IIO does not have to ability to create instruments or send orders or quotes to the market. Its sole responsibility it to send real time price updates on defined instruments to Actant Quote.

The External Price Feed IIO loads up a DLL written by Actant users. This DLL is the ClientModule which takes on the responsibility of providing prices for the instruments loaded. The External Price Feed IIO and its Client Module cannot exist on their own, an additional Actant IIO which makes a market connection is essential to complete the picture. Actant will not start up successfully without this.

SDK Installation

The SDK is delivered as a ZIP file called “ActantPricefeedSDK2.zip”.

This file should be unzipped into the “C:\Program Files\QT” directory.

If your Actant Quote installation is located elsewhere the installation will be more complex. The registry setup and batch files assume the default location.

Unzipping the SDK delivers the following:

- An IIO to the AQTOR directory (ExternalPriceFeedIIO.dll)
- Creates a directory PriceFeedDLLSDK
    - Two project files to build the demo in either vc7 or vc9
    - Two registry scripts to set up the registry to load the IIO and Demo DLL, both a vc7 and vc9 version
- Creates a sub directory of this with code of a demo ClientModule (DemoPriceFeedDLL)
- C files which implement the DLL entry points and API structures and enums (ExternalPriceFeedAPI.cpp/h)
- ReadMe.txt file

Build the DemoPriceFeedDLL project in either VC7 or VC9 then run the corresponding registry script. The two VC projects generate different executable file names and therefore require a different registry setup.

The IIO which loads the ClientModule DLL is a COM DLL loaded by actant at startup. It is delivered to the AQTOR directory when the SDK is unzipped. It must therefore be registered from the AQTOR directory using the following command:

```
C:\Program Files\QT\AQTOR>regsvr32 ExternalPriceFeedIIO.dll
```

A second actant IIO must also be enabled and configured. Since the ClientModule is only a price feed another IIO is required to provide full access to the market. The other IIO is responsible for defining the instruments and handling the sending of orders and quotes to the market.

IIOConfiguration

After running the registry setup script the External Price Feed IIO setup should appear in the registry. It has a sub directory Config which is displayed in the second window.

- **DemoPriceFeedDLL**

  The name of the registry key under the Interfaces key appears in the traffic light in the main actant window. Multiple such keys may be created here to load up multiple ClientModules if required

- **Enabled**

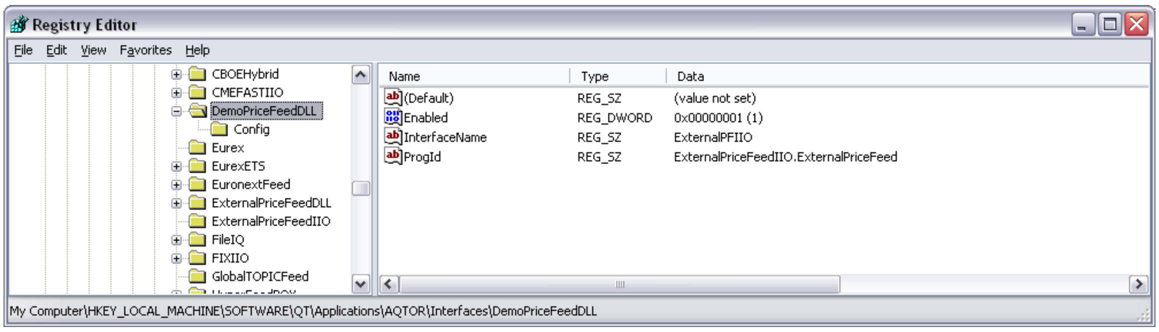
  Creates an IIO instance from the COM ProgId if set to 1. 0 implies disabled.

- **InterfaceName**

  The name of the interface used to look up parameters

- **ProgId**

  COM ProgId used to create the IIO which loads up the ClientModule. This must have the value "ExternalPriceFeedIIO.ExternalPriceFeed"



```
CBOEHybrid
CMEFASTIIO
DemoPriceFeedDLL
    Config
Eurex
EurexETS
EuronextFeed
ExternalPriceFeedDLL
ExternalPriceFeedIIO
FileIQ
FIXIIO
GlobalTOPICFeed
```

Name	Type	Data
(Default)	REG_SZ	(value not set)
Enabled	REG_DWORD	0x00000001 (1)
InterfaceName	REG_SZ	ExternalPFIIO
ProgId	REG_SZ	ExternalPriceFeedIIO.ExternalPriceFeed

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\QT\Applications\AQTOR\Interfaces\DemoPriceFeedDLL

- **Config**

  A sub key which contains the configuration for the ClientModule
  The developer may create their own configuration variables in here and access them courtesy of the registry location supplied at ClientModule startup.

- **PluginPriceFeedConfiguration**

  A string which can be set to supply the ClientModule configuration. If all the required configuration can be supplied in one simple string the ClientModule developer need not make further use of registry access functions. This string is passed into the ClientModule at startup.

- **PluginPriceFeedDLLName**

  The full path name to the DLL which is to be loaded by the External Price Feed IIO.

- **PriceFeedOutrights**

  A list of actant market codes for which the ClientModule is responsible for supplying outright BBO prices.

- **PriceFeedStrategies**

  A list of actant market codes for which the ClientModule is responsible for supplying both outright and strategy BBO prices.

- **PriceFeed**

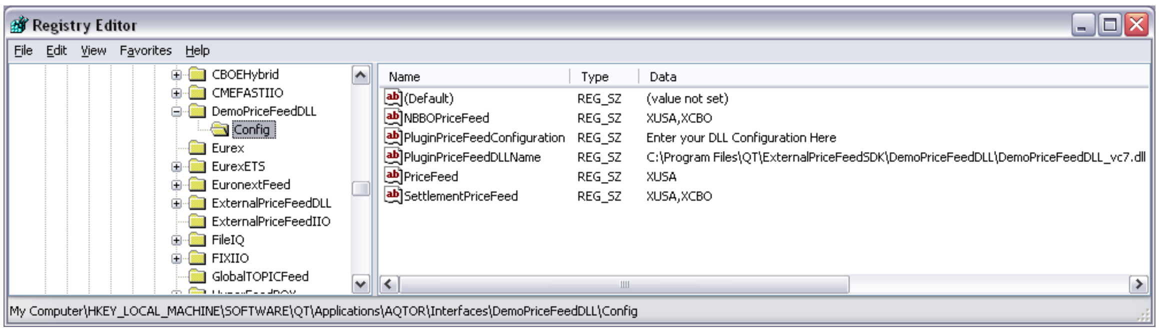
  A list of actant market codes for which the ClientModule is responsible for supplying BBO prices and other local instrument prices and values. This key is only supported for legacy purposes and is to be replaced by PriceFeedOutrights and PriceFeedStrategies.

- **NBBOPriceFeed**

  A list of actant market codes for which the ClientModule is responsible for supplying NBBO prices.

- **SettlementPriceFeed**

  A list of actant market codes for which the ClientModule is responsible for supplying Settlement prices.



Name	Type	Data
(Default)	REG_SZ	(value not set)
NBBOPriceFeed	REG_SZ	XUSA,XCBO
PluginPriceFeedConfiguration	REG_SZ	Enter your DLL Configuration Here
PluginPriceFeedDLLName	REG_SZ	C:\Program Files\QT\ExternalPriceFeedSDK\DemoPriceFeedDLL\DemoPriceFeedDLL_vc7.dll
PriceFeed	REG_SZ	XUSA
SettlementPriceFeed	REG_SZ	XUSA,XCBO

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\QT\Applications\AQTOR\Interfaces\DemoPriceFeedDLL\Config

The configuration of the IIO will determine which Subscription calls are made on the ClientModule.

The registry scripts included with the SDK set up the market codes to reflect those used at the CBOE, i.e. an underlying market code of XUSA and a derivative market code of XCBO. These will have to be changed if a connection to a market other that CBOE is being used.

The ClientModule DLL

The ClientModule is a windows DLL with a set of predefined entry points. The SDK contains a file called “ExternalPriceFeedAPI.cpp” and its associated header file “ExternalPriceFeedAPI.h” which should be compiled into the ClientModule DLL. This file implements the external functions and passes the calls on to typed functions which must be implemented by the developer. These functions are pre-implemented in this way to reduce the risk of type errors.

The API header file contains two namespaces, ClientModule and ClientCallback.

ClientModule namespace

The developer should implement each of the functions in the ClientModule namespace. These functions will receive calls via the DLL boundary from Actant. Each call is defined in the subsections below.

Startup

Called when the application is starting up. Two strings are passed in as parameters. The first is a configuration string loaded from the area of the registry which defines the ClientModule configuration. The second is the location of the registry from which the ClientModule is configured. The developer of the ClientModule has two choices for reading registry configuration, to either make use of the simple string or read their own variables. The approach taken will depend on the complexity of the configuration required.

The ClientModule should do all its checks to determine if startup can be performed successfully, read configuration files, load parameters etc before returning zero for success or non zero for failure. Returning an error code will prevent a complete application startup.

Shutdown

Always called on the ClientModule when the application is shutting down. Its a chance to delete memory, tidy up, stop threads etc.

Return zero for success, non zero for failure.

Connect

After a successful startup the ClientModule is asked to make its external connections to price sources.

Return zero for success non zero for failure. Returning an error code will prevent complete application startup.

Disconnect

Called at shutdown to ask the ClientModule to disconnect from markets/price sources.

Note if the ClientModule has previously reported connection loss this call is not made. Shutdown is always called.

SubscribeBBO

If the ClientModule has indicated via the registry that it will supply the BBO for one or more actant markets each time an instrument on such a market is opened i.e. put on display or used internally, the ClientModule will receive a call on SubscribeBBO. This function is passed an instrument structure detailing the instrument now requiring prices.

Once this call has been made for a particular instrument it is the responsibility of the ClientModule to provide the price updates on the instrument as they happen. Initially a full snapshot of the instrument’s prices should be passed to actant via the ClientCallback interface by making calls to the PriceFeed function.

The Client Module should send up all market data for a given instrument except the NBBO and settlement prices, which will be enabled through separate calls. This includes sending indicative market prices.

The call gets passed an Instrument structure that contains detailed instrument information. It can be used to match this information with the instrument identification of the external data source. Be careful when matching the strike price, doubles equality can fail due to rounding errors!

Some external APIs might require an exchange specific identifier to be used when communicating with the price feed service. Where applicable this info will be passed in a field called InstrumentExchangeId. *An example is for instance PHLX where instrument identifier specific to an option/strategy needs to be read from SQF API and supplied in a TOPO API*.

SubscribeNBBO

In a similar way the ClientModule can elect to supply NBBO prices for an instrument. If so this function is called to indicate these are now required. NBBO prices are sent to actant in a similar way to BBO prices.

SubscribeSettlement

Additionally the Settlement prices, yesterday’s close price, for instruments may be supplied by the client module.

Unsubscribe

Called when the instrument subscriptions are not longer required. All subscriptions should be turned off.

SubscibeStrategyBBO

This call has the same meaning as SubsctibeBBO. The main difference is the signature. To identify a strategy and match it with a definition in the external price source, the complete definition is passed to the ClientModule in ExtPF::Strategy.

Flipped Strategies

When matching ExtPF::Strategy with the strategies in the external data source, the definitions may be flipped. E.g. ExtPF::Strategy contains “BUY 1 ING.DEC22.185.C, SELL 1 ING.DEC22.190.C”, while the external data source would contain “*SELL* 1 ING.DEC22.185.C, *BUY* 1 ING.DEC22.190.C”. This has to be treated as a match, but the sign of all prices received from the external price source has to be changed. E.g. a bid of 1.25 received from the external price source has to be sent up as -1.25.

The InstrumentId field in ExtPF::Strategy and ExtPF::Instrument are unique in the scope of an IIO. No ExtPF::Strategy and ExtPF::Instrument will ever get the same InstrumentId. This allows the ClientModule to handle instrument and strategy subscriptions with the same code. For that reason the callback API is the same for strategies and instruments.

There are no subscription calls for NBBO and Settlement price.

UnsubscribeStrategy

Called when the strategy subscription is not longer required.

SubscribeMarketStatus

The ClientModule must send the current market status of the given instrument immediately and then start updating the external price feed IIO with any status changes that occur. This is done with the MarketStatus callback.

UnsubscribeMarketStatus

The ClientModule should no longer send market status updates for the given instrument.

SubscribeStrategyMarketStatus, UnsubscribeStrategyMarketStatus

Same behaviour as SubscribeMarketStatus and UnsubscribeMarketStatus, but for strategies.

ClientCallback namespace

The ClientCallback namespace contains a number of functions implemented by the API code. These functions pass back calls to actant to supply market data, logging and status info. The following entry points are supplied.

LogMsg

Called by the ClientModule to log details out to the actant log file and the application logger window. The logCategory indicates the type of message and where it is to be delivered:

- Error
  Delivered to both the logger and the GUI. Errors are displayed in red
- Warning
  Delivered to both the logger and the GUI. Warnings are displayed in orange
- InfoToFile
  An information message which is written to the actant log file only, it does not appear in the GUI
- InfoToGUI
  An information message written to both the GUI and the Log file.

The other parameters BaseId and Message are the strings to be output. The BaseId can either be the Instrument Base Id or when the message is not instrument related it can suggest the general location of the messages source.

Both these strings are output to the GUI for GUI messages and the logger file for logger messages.

PriceFeed

The ClientModule delivers instrument and strategy market data snapshots and increments via calls to this function.

There are a number of types of update listed in the ExtPF::UpdateType enum. These update types represent the meaning of the data being delivered i.e. the attribute being changed. There are three delivery mechanisms for priced data:

- Price Only
  Some update types deliver only prices. An example of this would be High or Low prices which do not have corresponding quantities. These updates are delivered in a ExtPF::PriceUpdateEntry.
- Quantity Only
  Likewise some updates such as total Traded volume have just a quantity without an associated price. These updates are delivered in ExtPF::QuantityUpdateEntry.
- Price And Quantity
  When a price and quantity are associated as in Last Price and quantity update (representing a trade) or Bid and Ask updates with their quantities, these are delivered in the ExtPF::PriceQuoteUpdateEntry structure.
  The iMarket field is used to differentiate between BBO, NBBO and indicative BBO price

updates. Please use the constants defined with the ExtPF::MarketSelection enum.

An array of each of the structure types may be passed to the PriceFeed call. The more data that can be supplied with each call the better as the updates are all processed in one transaction though actant. Likewise if a price and a quantity can be tied together in a ExtPF::PriceQuoteUpdateEntry this has advantages over delivering both a ExtPF::PriceUpdateEntry and a ExtPF::QuantityUpdateEntry each with the same update type.

Each call to PriceFeed relates to one instrument only specified by the instrumentId parameter. This value can be found in the ExtPF::Instrument or ExtPF::Strategy structure passed in the original subscription calls.

For performance reasons it is recommended that the ClientModule keeps arrays of ExtPF::PriceUpdateEntry, ExtPF::QuantityUpdateEntry and ExtPF::PriceQuoteUpdateEntry in memory and doesn’t allocate them for each update. Market data updates should be passed on to the external price feed IIO without any heap allocations.

MarketStatus

The ClientModule delivers market status snapshots and updates for instruments and strategies through this call.

There is a final list of market states defined in ExtPF::MarketStatus that can be used. If a market state required on the exchange isn’t present in the list, please contact Actant. Sending other integer values outside the definition of the enum in the API will cause an error message in the logger window.

ConnectionLoss

If connectivity to the price source is lost at some point after startup the ClientModule must inform actant by calling the ConnectionLoss function. Once called the connection cannot be recovered without an application restart.

ConnectionUnreliable

If the ClientModule detects problems with the price feed which would imply the price data is being delayed the ConnectionUnreliable call can be used to alert actant. This causes any quotes in the market which are dependant on the price feed to be pulled since the quote prices may be out of date.

Situations where this is useful may be when the queue of pending price updates grows to a certain size or when the latency between the market and the application grows to large figure.

Pulled quotes may be re-entered by the user as they see fit.

US market codes

Here is a list of the market codes that Actant uses for US markets. For actant internal reasons Underlyings for one derivative market are treated as seperate from underlyings for another market. These codes are used in the registry configuration for the client price feeds to specifiy for which markets the price feed can deliver prices (See chapter 4).

Market	**Derivate market code**	**Underlying market code**
ARCA	XPCX	XUSP
AMEX	XAMX	XUSA
CBOE	XCBO	XUSA
CBOE C2	XC2O	XUS2
PHLX	XPHL	XUSL
ISE	XISE	XUSI
CME	XCME	

Client Module DLL

External Price Feed IIO

Market

External Data Source

Q

U

O

T

E

Market

External

Data Source

Quote IIOs