Tradesignal OpenConnect SDK 6.2

Published: 07.03.2014

Introduction

Organizations increasingly require the advanced charting and analysis capabilities of Tradesignal products to run against proprietary in-house data sources.

Tradesignal OpenConnect is a Windows service that sits between the proprietary data source and the DataConnect Service; regulating data flow from one to the other. The creation of an Adapter Module that Tradesignal OpenConnect uses to communicate with the upstream data source is all that is required to facilitate such data access. Creation of a new Adapter Module for a familiar data source is anticipated to take between two and four weeks.

This document describes the Tradesignal OpenConnect framework and the necessary tasks an Adapter Module must perform in order to successfully integrate with the framework.

Table of Contents

The architecture of DataConnect with Tradesignal OpenConnect

The typical deployment architecture consists of one or more Tradesignal clients connecting to a DataConnect service. The DataConnect service receives its data from the OpenConnect service, which acts as a proxy to the Proprietary Data Server via the Adapter Module.

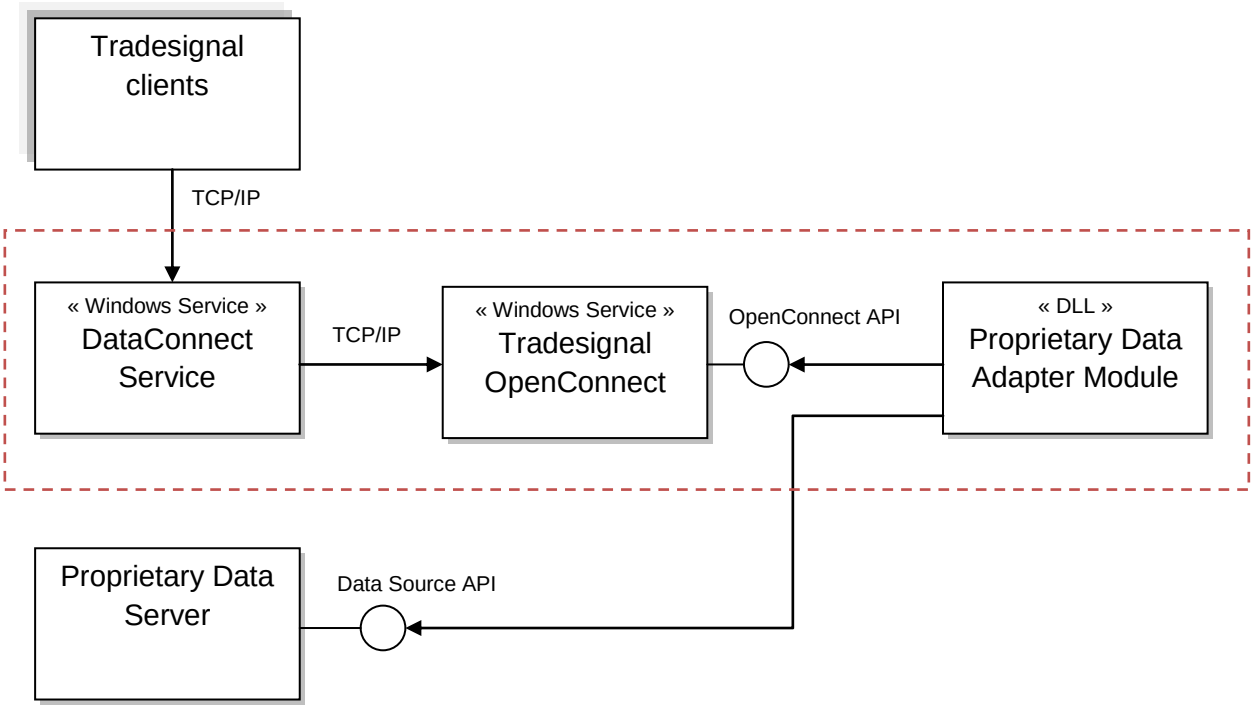

Figure 1 – Typical deployment of an OpenConnect service. The dashed box denotes a single physical machine (OpenConnect could alternatively be deployed on a separate machine to DataConnect)

The main components are:

Tradesignal DataConnect Service – Service responsible for managing multiple client connections, collecting historic data in an optimized format for fast delivery, disseminating real-time quotes, client licensing and price corrections.

Tradesignal OpenConnect – Middleware that provides an open API allowing a DLL to be created that facilitates the connection between DataConnect and a proprietary data source.

Adapter Module – DLL created specifically to implement the OpenConnect API and complete the various requests by utilizing the interface of a proprietary data source.

Proprietary Data Server – The server, database or feed from which Tradesignal must ultimately obtain data from.

Contents of the Tradesignal OpenConnect SDK

The Tradesignal OpenConnect SDK setup will install the following items to the hard drive of a development machine.

Item	Description
Tradesignal OpenConnect SDK.pdf	This document.
Bin/OpenConnect.exe	The redistributable OpenConnect middleware.
Bin/OpenConnect.tlb	COM Type library for the OpenConnect API.
Bin/OpenConnect.Interop.dll	Redistributable .NET Wrapper to the COM Type library.
Samples/C#/TickPump	Sample Visual Studio 2010 project that builds an Adapter Module that will push random ticks to DataConnect via OpenConnect.
Samples/C#/CSVFileHandler	Sample Visual Studio 2010 project that builds an Adapter Module that will monitor folders on the file system for CSV formatted data files and deliver them as time series. Changes to the files will be delivered to DataConnect as new ticks/data corrections.

Building and running the sample projects

The two sample projects can be opened in Microsoft Visual Studio 2010 (or later versions) and built in the normal way. The projects will automatically register the built Adapter Module and copy the OpenConnect.exe file from the *bin* directory to the build directory.

Sample projects (once built) can be run through the debugger as follows:

1. Edit the sample project settings (right-click on the project in Solution Explorer and select *Properties*).
2. Click on the *Debug* tab in the project settings window and under *Start Action*, select *Start External Program*.
3. Enter the location of the OpenConnect.exe that has been copied to the build directory in the field next to *Start External Program*.
4. Under *Start Options* in the field *Command line arguments* enter the switch "*-d"*.
5. Select *Debug* > *Start Debugging* on the main menu to run the sample in debug mode.

Connecting DataConnect to an OpenConnect server

To connect a TDMS to an OpenConnect server the following steps are necessary:

- Install DataConnect as normal (*DataConnect 3.6* or greater must be used)
- During installation, select *OpenConnect* as the *Infrastructure*.
- When prompted, enter the service name that has been chosen for the OpenConnect service (e.g. for the TickPump sample project one would enter *SampleTickPump*). When using DataConnect 5 or above, select a (non-empty) prefix used to identify symbols served by this feed.
- When prompted, enter the machine on which the OpenConnect service is running (normally *localhost*).
- Obtain a license code for the use of OpenConnect with DataConnect (from the Tradesignal support team) and register it with the DataConnect license manager.
- Ensure the OpenConnect service is running and restart the DataConnect service.

The Tradesignal OpenConnect service, Adapter Modules and Deployment

An Adapter Module is an in-proc free-threaded COM server containing a CoClass that implements the required OpenConnect adapter module interface. The Class ID of the entry point CoClass is specified in an *.ini* file that the OpenConnect service reads during start-up.

Deploying an Adapter Module

It is expected that a typical distribution of an OpenConnect service will consist of the following files, located in a single directory on the deployment server:

File	Description
OpenConnect.exe	The windows service.
OpenConnect.ini	The initialization file that configures OpenConnect and can be accessed from the adapter module via COM interfaces.
OpenConnect.Interop.dll	.NET helper library that must be deployed with OpenConnect ***only*** if the adapter module is a .NET assembly.
C++ v10 Runtime	C++ dependency of OpenConnect.exe, see note below.
<adaptermodule>.dll *	The bespoke adapter module COM Server.
<other files>	Any files needed by the adapter module.

* COM servers must be registered on the deployment system, if it is a C/C++ COM DLL then *regsvr32.exe* should be used for registration, if it is a .NET COM DLL then use *regasm.exe*.

Important Note: *OpenConnect.exe* is dependent on the C++ runtime version 10.0.40219.1. The deployment mechanism must ensure that all necessary DLLs are present on the deployment machine. It is recommended to include the merge module *Microsoft_VC100_CRT_x86.msm* when creating a setup for deployment.

OpenConnect.exe

OpenConnect.exe supports the following command line arguments:

Switch	Description
-i	Install as a windows service
-s	Starts the windows service
-u	Uninstall the windows service
-d	Start in debug mode

Please ensure you are using administrator rights to install and start the service. Once the service has been installed, the Windows Service Control Manager (SCM) should be used from then on to start and stop it.

If *OpenConnect.exe* is run with the ***-d*** command line flag, it will operate like any normally run application (without the background execution or automatic restart behavior of running under the SCM). Running outside of the SCM is a useful feature for debugging while developing an Adapter Module.

It is possible to register multiple instances of *OpenConnect.exe* on a single machine, so long as: they are in different directories; they use different Adapter Modules with unique service names and COM Class IDs; and the *OpenConnect.ini* files specify different ports.

OpenConnect.ini

The initialization file for OpenConnect is used to identify the Adapter Module to be used. The OpenConnect.ini file must be an ANSI encoded text file present in the same directory as *OpenConnect.exe*. On start-up, *OpenConnect.exe* will look for the following setting:

```
[General]
AdapterModuleClassId = <class id GUID> ; e.g. {6C4D8623-C999-4180-A0F6-F986F610165B}
```

Upon finding the setting, OpenConnect will attempt to create an instance of the CoClass with that Class ID. This CoClass must implement the interface *IOpenConnectAdapterModule* found in the Type Library associated with *OpenConnect.exe*.

There are a number of other settings that may be specified in OpenConnect.ini although its main function is to consolidate all settings for both OpenConnect.exe and the Adapter Module. Any setting in the file can be read by the Adapter Module using the specific *IConnectionRequest.Get<type>Setting()* methods. See *Appendix A – OpenConnect.ini Settings* for a description of the OpenConnect.ini file settings.

When starting OpenConnect.exe with the –d option (e.g. for debugging purposes), it might be helpful to set the following configuration option that instructs OpenConnect to mirror all output written to the log file also to the Console window:

```
[Internal]
StandardOutput=1
```

OpenConnect.log

Messages from both OpenConnect.exe and an Adapter Module will be written to the *OpenConnect.log* file located in the same directory as *OpenConnect.exe*. Certain types of messages will additionally be written to the Windows Event Log so they may be picked up by third-party monitoring software.

The various levels of message and where they will be sent are determined by the *ELogLevel* enumeration. An Adapter Module can send its own messages to the log system using the *IConnectionRequest.WriteLogMessage()* method.

Writing too many messages into the log system may introduce performance issues. It is recommended to make use of the *ELogLevel.LogDebug* message type to ensure detailed logging is present, but must first be enabled using the configuration setting:

```
[Log]
LogDebug=1
```

It should also be noted that use of the *ELogLevel.LogImportant* message type should be restricted to very significant informational messages that occur rather infrequently. Such messages are written to the Windows event log and generation of too many messages will obscure important information. The best use of this level is to indicate start-up, successful connection to a feed, shutdown, etc.

Creating an Adapter Module

An Adapter Module is an in-proc COM server located in a DLL. The Adapter Module must contain a creatable CoClass that implements the *IOpenConnectAdapterModule* interface that is found in the type library of OpenConnect; see the file OpenConnect.tlb in the SDK *bin* directory. For .NET projects the OpenConnect.Interop.dll file should be added as a project reference in order to access the OpenConnect type library. The type library contains all other interfaces that are necessary to successfully implement an Adapter Module.

OpenConnect uses the free-threaded COM model; methods from the OpenConnect interface can be safely called from many threads.

HRESULT return types

Unless otherwise indicated, API methods will only return the following values; specific details will be sent to the log file. An Adapter Module is also expected to follow these conventions.

HRESULT	Description
S_OK	Method succeeded
E_INVALIDARG	An invalid argument was provided
E_POINTER	A null pointer argument was provided
E_OUTOFMEMORY	Ran out of memory
E_FAIL	Other failure cases (no connection, etc.)

Security Lists

Some types of requests require a result to be sent via the *SendListAsXml()* method of the *ISecurityListRequest* interface. The expected XML responses are as described below.

OpenConnect supports arbitrary nested, hierarchic security list catalogues. Up to version 5.5, the only way to provide the security list catalogue was to publish it completely in one go (see below, requested through *IOpenConnectAdapterModule.OnRequestSecurityList()*). While this approach fits many use cases nicely, it may not be well adapted to very large and complex hierarchies or in the case where requesting the full catalogue at once from the upstream data feed takes prohibitively long. Therefore OpenConnect offers an alternative way to publish the security list catalogue in a drill-down, on-demand fashion. For more information please refer to the section called ”On-demand security catalogue folder retrieval”.

Security list catalogue format (full catalogue request)

The below XML schema and following XML example, describe the expected result from a security catalogue request. Below the document element, either folders or list elements may be present. A folder element may additionally contain lists or additional folders (to an arbitrary depth).

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="security-list-catalogue">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="folder" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="version" type="xs:decimal" use="required" />
    </xs:complexType>
  </xs:element>
```

```
  <xs:element name="folder">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="folder" minOccurs="0" maxOccurs="unbounded" />
        <xs:element maxOccurs="unbounded" name="list">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="lid" type="xs:string" use="required" />
                <xs:attribute name="type" type="xs:string" use="optional" />
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The *lid* value is the same value that will be supplied with the *IOpenConnectAdapterModule.OnRequestSecurityList()* method to identify a list.

The type attribute should be from the following list (unknown or empty types will be ignored): *index*, *stock*, *future*, *rolling-future*, *bond*, *option*, *warrant*, *certificate*, *currency*, *other*.

Example security list catalogue:

```
<?xml version="1.0" ?>
<security-list-catalogue version="1.0">
  <folder name="Stocks">
    <folder name="Germany">
      <list lid="DAX30">DAX 30</list>
      <list lid="MDAX">MDAX</list>
    </folder>
    <list lid="FTSE">FTSE</list>
    <list lid="INDU">Dow Jones Industrial Average</list>
  </folder>
  <folder name="Futures">
    <list lid="FDAX" type="future" >Dax Future Contracts</list>
    <list lid="S" type="future" >Soybeans Future Contracts</list>
  </folder>
</security-list-catalogue>
```

Security list format

The below XML schema and following XML example, describe the expected result from either a security list request, or a search request. All attributes with the exception of *sid* are optional, although if present will allow more client-side sub-searching and grouping possibilities.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="security-list">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="item">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="sid" type="xs:string" use="required" />
                <xs:attribute name="ticker" type="xs:string" use="optional" />
                <xs:attribute name="type" type="xs:string" use="optional" />
                <xs:attribute name="isin" type="xs:string" use="optional" />
                <xs:attribute name="contractMonth" type="xs:decimal" use="optional" />
                <xs:attribute name="contractYear" type="xs:decimal" use="optional" />
                <xs:attribute name="rootCode" type="xs:string" use="optional" />
              </xs:extension>
```

```
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" type="xs:decimal" use="required" />
      <xs:attribute name="lid" type="xs:string" use="optional" />
      <xs:attribute name="name" type="xs:string" use="optional" />
      <xs:attribute name="type" type="xs:string" use="optional" />
      <xs:attribute name="contractperiod" type="xs:string" use="optional" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The *sid* value is the same value that will be supplied with the *IOpenConnectAdapterModule.OnSubscribeSecurity()* method to identify a security.

The type attribute should be from the following list (unknown or empty types will be ignored): *index*, *stock*, *future*, *rolling-future*, *bond*, *option*, *warrant*, *certificate*, *currency*, *other*.

The contract period attribute can be specified as an optional hint for the trading period of futures contracts and should be one of the following strings: *days, weeks, months, quarters, years.*

The security list item attributes contract month, contract year and root code should be specified for futures contracts, containing expiry and future root code information.

Example symbol list for stocks:

```
<?xml version="1.0" ?>
<security-list version="1.1" lid="DAX2" name="DAX 2 Composite">
  <item sid="SIEGn.DE" ticker="SIE" type="stock" isin="DE0007236101">Siemens</item>
  <item sid="DTEGn.DE" ticker="DTE" type="stock" isin="DE0005557508">Deutsche Telekom</item>
</security-list>
```

Example symbol list for futures:

```
<?xml version="1.0" ?>
<security-list version="1.1" lid="FutFDAX" name="EUREX DAX Futures" contractperiod="months">
  <item sid="FDXH3" type="future" rootCode="FDX" contractMonth="3" contractYear="2013">
   DAX Future March 2013
  </item>
  <item sid="FDXM3" type="future" rootCode="FDX" contractMonth="6" contractYear="2013">
   DAX Future June 2013
  </item>
  <item sid="FDXU3" type="future" rootCode="FDX" contractMonth="7" contractYear="2013">
   DAX Future September 2013
  </item>
  <item sid="FDXZ3" type="future" rootCode="FDX" contractMonth="12" contractYear="2013">
   DAX Future December 2013
  </item>
</security-list>
```

On-demand security catalogue folder retrieval (drill-down request)

Alternative to the full catalogue retrieval, DataConnect can be instructed to retrieve catalogue nodes on-demand. This means that when a Tradesignal user navigates through the symbol catalogue, the contents of every opened folder will be retrieved on folder-by-folder basis, instead of using a cached version of the full security list catalogue.

OpenConnect adapter modules will have to support one of the two approaches, but there is no need to support both. Requests for on-demand catalogue folders will only then be issued

by DataConnect, when the adapter module implements the *IOnDemandSecurityCatalogue* interface. In that case, *IOpenConnectAdapterModule.OnRequestSecurityList()* will never be fired.

Requests for a specific folder in the catalogue hierarchy will be forwarded via *IOnDemandSecurityCatalogue.OnRequestFolder()*, passing in a string uniquely identifying the folder in question. By convention, the top-most folder will be identified through the empty string. The unique identifier for all sub-nodes is arbitrarily assignable by the OpenConnect module implementer.

Example security list catalogue hierarchy (underlined names denote symbol lists):

```
Stocks
        Germany
                DAX 30
                        (ADS, ALV, BAS, …)

        USA
                NYSE
                        (ABT, ACN, AGN, …)
                NASDAQ
                        (AAPL, ADBE, ADP, …)
Futures
        USA
                Commodities
                        Soy
                                (SZ2, SH3, SM3, …)
                        Wheat
                                (WZ2, WH3, WM3, …)
```

The top-folder request (id = “”) would then return an Xml response similar to the following example (to be sent via the passed in *ISecurityListRequest* interface):

```
<?xml version="1.0" ?>
<folder fid="">
  <folder name="Stocks" fid="stocks" />
  <folder name="Futures" fid="futures" />
</folder>
```

The values supplied in the fid attribute of the inner folder nodes can be arbitrarily chosen by the OpenConnect adapter module implementer (as long as they resemble valid Xml attribute strings).

A drill-down request for the contents of the “Stocks” folder (via the folder id “`stocks`”) would yield this:

```
<?xml version="1.0" ?>
<folder fid="stocks">
  <folder name="Germany" fid="stocks|germany" />
  <folder name="USA" fid="futures|usa" />
</folder>
```

A further drill-down request for the contents of the “Germany” folder inside the “Stocks” folder (via the folder id “`stocks|germany`”) would yield this (note the usage of the `<list>` node and `lid` attribute):

```
<?xml version="1.0" ?>
<folder fid="stocks|germany">
  <list name="DAX 30" lid="chain:dax30" />
</folder>
```

Drilling down into groups of unknown contents

For some upstream data feeds it may be difficult to determine the type of a security catalogue group beforehand, i.e. a folder or a terminal list without actually populating the contents of the node. The OpenConnect API therefore allows to return `<unknown>` nodes instead of `<folder>` or `<list>` to enable fast response times and therefore a better user experience:

```
<?xml version="1.0" ?>
<folder fid="myId">
  <unknown name="DAX 30" id="chain:dax30" />
</folder>
```

The determination of the actual type is then delayed until the Tradesignal user actually opens a specific node. The request for the type of a node will be forwarded through *IOnDemandSecurityCatalogue.OnRequestGroupType()*, passing in the id of the group in question. The response has to be published via the *ISecurityListRequest* interface:

```
<?xml version="1.0" ?>
<unknown id="chain:dax30">
  <list name="DAX 30" lid="chain:dax30" />
</folder>
```

In case the requested group is actually a folder, the response would look like this:

```
<?xml version="1.0" ?>
<unknown id="stocks|germany">
  <folder name="Germany" fid="stocks|germany" />
</folder>
```

Common coding mistakes and gotchas

Ensuring unique time stamps

DataConnect requires that the feed handler supplies unique time stamps for bars and ticks. This is essentially for two reasons:

1. It guarantees that there are no data ordering issues because the feed handler author is forced to consider this issue
2. DataConnect requires a completely deterministic way to inform the feed handler of the last tick that it received even when a feed handler has no support for unique trade IDs

For handling time stamps with millisecond granularity in C++, see *Appendix C – Passing time stamps in interface methods when using* C++.

Ensuring asynchronous responses to DataConnect requests

Most of the callbacks that must be implemented as part of the `IOpenConnectAdapterModule` expect prompt returns and that the supplied interface is used some time later to answer the original request. Such behavior almost certainly requires a separate thread to be managing the response interfaces.

Blocking inside the request callbacks, for example when waiting for a database operation to complete, will normally result in a timeout message in DataConnect. Please favour doing database queries in a background thread; all response interfaces are free-threaded and so can be used safely between multiple threads.

Supporting creation of forward curves and seasonal charts

Forward curves and seasonal charts in Tradesignal are created and stored based on symbol lists. In order to support creation of those chart types, the symbol lists must be tagged as containing future symbols (type="future") and provide the fields: *type*, *rootCode*, *contractMonth* and *contractYear* for each symbol in a list.

Additionally, it is necessary to allow Tradesignal clients to search for a specific symbol list. This will be done by a normal symbol list request. The *list id* requested will then be a root code or a symbol name post-fixed by either *~rel* (for active contracts) or *~exp* (for expired contracts) to indicate what type of list is currently requested. The adapter module has to correctly resolve those requests to real symbol lists of the appropriate type. For example:

FX~rel – A search for a related symbol list, this would expect to get a symbol list back containing all currently active symbols related to the given root code: *FX*. The correct name of that symbol list has to be supplied in the *lid* attribute of the resulting symbol list.

FX~exp – A search for a related symbol list containing expired contracts related to the given root code: *FX*. The correct name of that symbol list should be supplied in the *lid* attribute of the symbol resulting list.

Important: Symbol lists containing the currently active contracts (*~rel*) have to be sorted ascending by their expiry date. Expired symbol list (*~exp*) have to be sorted descending.

Tradesignal also allows creating forward curves directly from an Equilla Script, passing in a symbol code that is used as the basis for the forward curve computation. To enable this functionality, the symbol must supply root code information. This can be achieved by specifying a contract's root code in the meta field data basis through the `IMetaField2` interface and the `SetStringField` function, providing *"*`OC_ROOT_CODE`*"* as the field identifier and the actual root code as the value.

Supporting rolling forward symbols (User-Defined Continuation Symbols)

Since DataConnect 5.2.0, it is possible to create user-defined continuation symbols (UDCs) in DataConnect. Those symbols will be created based on root codes identifying the underlying future symbols and two matching symbol lists providing the current and the expired symbols of the base root code (see the previous section for an explanation on naming such lists and how they will be requested).

DataConnect will supply a related and an expired search for the root code given by the user. The UDC will get successfully created if the symbol list requests succeed and the DataConnect gets valid *list ids* back. Based on these lists the UDC symbol will be created.

Important: Symbol lists containing the currently active contracts have to be sorted ascending by their expiry date. Expired symbol list have to be sorted descending.

Collecting Data vs On-demand Data

OpenConnect can be configured to not collect and store data in DataConnect, but instead to delegate all historic data requests in all periods to the Adapter Module. This mode is activated by setting the *PersistentSymbols* property in the *General* section of *OpenConnect.ini* to 0.

The on-demand data mode is most useful where the upstream data source already stores pre-cumulated data (minute, hourly, daily, etc.) and represents the only location where data corrections and the like can occur (assuming there is no mechanism to forward them to DataConnect).

Please note: Care must be taken with the on-demand mode that the adapter module and upstream databases can cope with the additional load a group of users will incur when the DataConnect is no longer responsible for storage of (and fast access to) the data.

OpenConnect COM Interfaces

The interfaces used by an Adapter Module can be seen in the following class diagram. For a full description of each interface, enumeration and method, please refer to the next section.

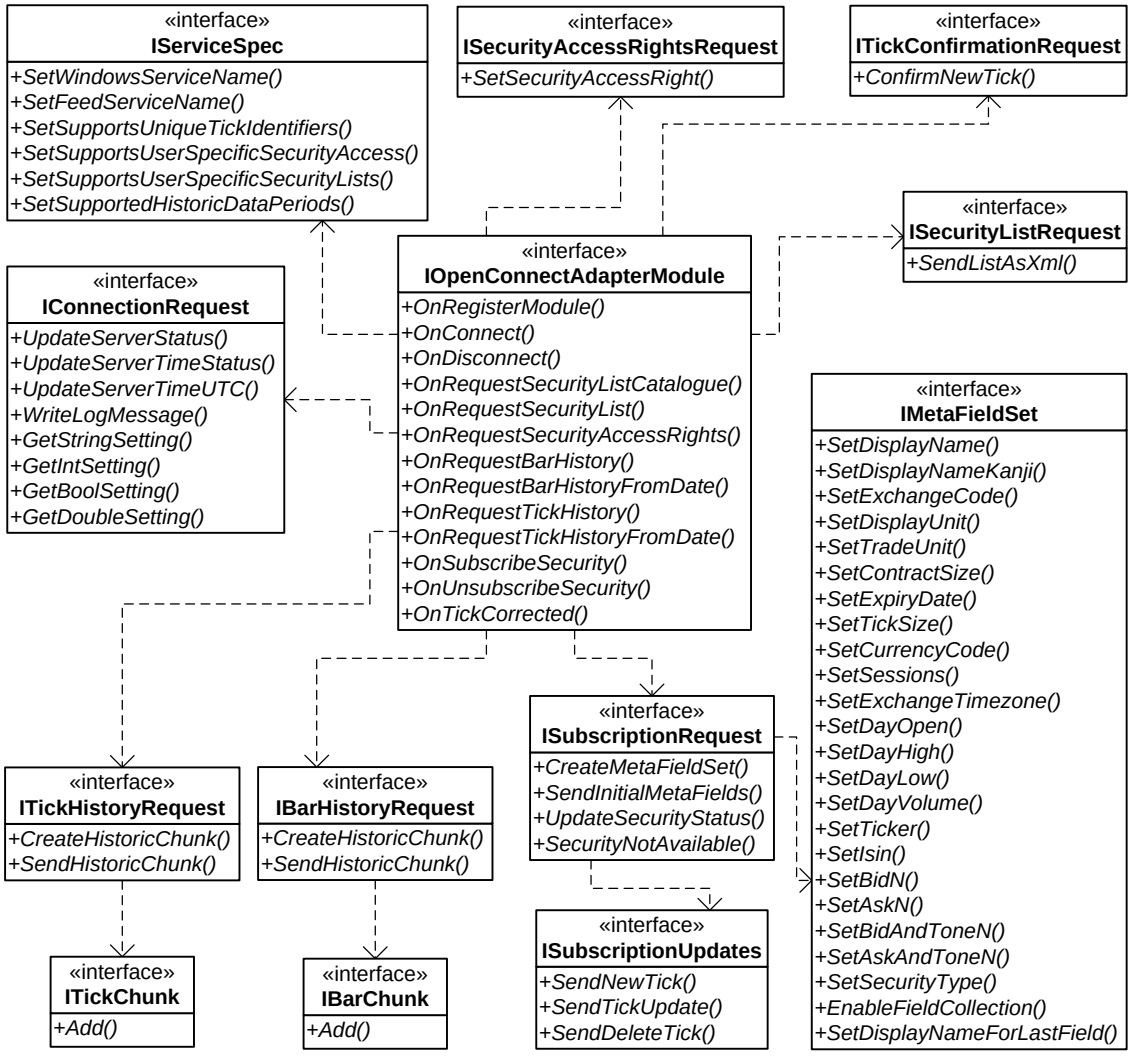

Figure 2 – OpenConnect framework interfaces

Application Programmers Interface (API)

The Tradesignal OpenConnect library provides the following interfaces for use by implementers to create an Adapter Module that will regulate the communication between OpenConnect and a third-party database or data feed.

IOpenConnectAdapterModule : Dispatch

The Adapter Module must define a Creatable CoClass that implements this interface to handle requests from the OpenConnect service. The CLSID (GUID) of that CoClass must be saved in OpenConnect.ini as '[General]/AdapterModuleClassId'.

OnRegisterModule

```
OnRegisterModule( «in» serviceSpec : IServiceSpec* ) : void
```

Called shortly after the DLL is attached to query parameters concerning this service.

OnConnect

```
OnConnect( «in» connection : IConnectionRequest* ) : void
```

Called to instruct the Adapter module to connect to the data source and send server time.

OnDisconnect

```
OnDisconnect( ) : void
```

Called to instruct the Adapter Module to close the connection.

OnRequestSecurityListCatalogue

```
OnRequestSecurityListCatalogue( «in» user : string,
                                «in» listRequest : ISecurityListRequest* ) : void
```

Called to obtain the catalogue of security lists for a given user, or all users if the user parameter is NULL.

OnRequestSecurityList

```
OnRequestSecurityList( «in» user : string, «in» list : string,
                       «in» listRequest : ISecurityListRequest* ) : void
```

Called to obtain a security list for a given user or all users if the user parameter is NULL.

OnRequestSecurityAccessRights

```
OnRequestSecurityAccessRights( «in» user : string, «in» security : string,
                       «in» rightsRequest : ISecurityAccessRightsRequest* ) : void
```

Called to query if the selected user is allowed to access the specified security.

OnRequestBarHistory

```
OnRequestBarHistory( «in» security : string, «in» field : EField,
                     «in» period : EDataPeriod,
                     «in» historyRequest : IBarHistoryRequest* ) : void
```

Called to obtain all the historic data for a given security/field pair in a specified period. The history should be sent in chunks ordered from newest to oldest data.

OnRequestBarHistoryFromDate

```
OnRequestBarHistoryFromDate( «in» security : string, «in» field : EField,
                             «in» period : EDataPeriod,
                             «in» historyRequest : IBarHistoryRequest*,
                             «in» fromDateTime : DATE ) : void
```

Called to obtain the historic data for a given security/field pair in a specified period from a given date time. The history should be sent in chunks ordered from newest to oldest data.

OnRequestTickHistory

```
OnRequestTickHistory( «in» security : string, «in» field : EField,
                      «in» historyRequest : ITickHistoryRequest* ) : void
```

Called to obtain all the historic tick data for a given security/field pair. The history should be sent in chunks ordered from newest to oldest data.

OnRequestTickHistoryFromDate

```
OnRequestTickHistoryFromDate( «in» security : string, «in» field : EField,
                              «in» historyRequest : ITickHistoryRequest*,
                              «in» fromDateTime : DATE ) : void
```

Called to obtain the historic tick data for a given security/field pair from a given date time. The history should be sent in chunks ordered from newest to oldest data.

OnSubscribeSecurity

```
OnSubscribeSecurity( «in» security : string,
                     «in» subscriptionRequest : ISubscriptionRequest* ) : void
```

Called to request the current data and updates/corrections as they occur for a given security. Note that security identifiers must not contain whitespace characters and must not exceed 100 characters in length.

OnUnsubscribeSecurity

```
OnUnsubscribeSecurity( «in» security : string ) : void
```

Called to request updates to a subscribed security be stopped.

OnTickCorrected

```
OnTickCorrected( «in» security : string, «in» field : EField,
                 «in» correctionType : ECorrectionType, «in» timestamp : DATE,
                 «in» value : double, «in» size : double, «in» tickId : int64,
                 «in» confirmationRequest : ITickConfirmationRequest* ) : void
```

Called to inform the Adapter Module on a manual tick correction occurred in DataConnect. The upstream server can then be updated to reflect this change, and optionally a new ID for the tick stored in DataConnect can be returned via the *confirmationRequest* interface.

IOpenConnectAdapterModule2 : IOpenConnectAdapterModule

This interface was introduced with the OpenConnect 5.7 SDK. It expands the *IOpenConnectAdapterModule* interface with methods that allow more fine grained historic data requests, potentially reducing the total data that needs to be transferred by a significant amount. In addition to that the Online/Offline methods provide more detailed information about the state of connectivity for the module implementer.

OnRequestLimitedBarHistory

```
OnRequestLimitedBarHistory ( «in» security : string, «in» field : EField,
                             «in» period : EDataPeriod,
                             «in» historyRequest : IBarHistoryRequest*,
                             «in» fromDateTime : DATE,
                             «in» toDateTime : DATE,
                             «in» maxBars : int64) : void
```

Called to obtain the historic data for a given security/field pair in a specified period from a given timestamp (*fromDateTime*) to an end timestamp (*toDateTime*). The amount of bars provided via the *historyRequest* interface should not exceed the value specified by the *maxBars* parameter. The history should be sent in chunks ordered from newest to oldest data. If more data than *maxBars* is available the oldest bars should be omitted.

DataConnect will use *OnRequestLimitedBarHistory* in favour of *OnRequestBarHistory* when the adapter module implements *IOpenConnectAdapterModule2*.

OnRequestLimitedTickHistory

```
OnRequestTickHistoryFromDate( «in» security : string, «in» field : EField,
                              «in» historyRequest : ITickHistoryRequest*,
                              «in» fromDateTime : DATE,
                              «in» toDateTime : DATE,
                              «in» maxTicks : int64) : void
```

Called to obtain the historic tick data for a given security/field pair from a given timestamp (*fromDateTime*) to an end timestamp (*toDateTime*). The amount of ticks provided via the *historyRequest* interface should not exceed the value specified by the *maxTicks* parameter. The history should be sent in chunks ordered from newest to oldest data. If more data than *maxTicks* is available the oldest ticks should be omitted.

DataConnect will use *OnRequestLimitedTickHistory* in favour of *OnRequestTickHistory* when the adapter module implements *IOpenConnectAdapterModule2*.

OnDataConnectOnline

```
OnDataConnectOnline() : void
```

This is called to notify that a DataConnect connection has been established. In case the OpenConnect feed is not using persistent symbols (pass-through mode) this notification will be sent once the *first* client (e.g. Tradesignal or DataConnect Console) connects.

OnDataConnectOffline

```
OnDataConnectOffline() : void
```

This is called when the DataConnect connection is closed. If the OpenConnect feed is not using persistent symbols (pass-through mode) this notification will be sent when the *last* client (e.g. Tradesignal or DataConnect Console) disconnects.

OnResolveLevel2Symbol

```
OnResolveLevel2Symbol( «in» security : string, «in» user : string,
                       «in» level2Request : IResolveLevel2SymbolRequest) : void
```

The DataConnect server does not collect Level2 data but only forwards that data if explicitly requested by a connected client. Supplying Level2 data for every subscribed instrument via the *IMetaFieldSet* will therefore be unnecessary (and often inefficient) when no client has explicitly shown interest to that kind of data. In those cases the delivery of said data can be redirected to a dedicated symbol.

IOpenConnectAdapterModule3 : IOpenConnectAdapterModule2

This interface was introduced with the OpenConnect 5.9 SDK. It expands the *IOpenConnectAdapterModule* interface with the ability to be notified when the underlying configuration file is changed.

OnSettingsUpdated

```
OnSettingsUpdated() : void
```

This method is called when a change in the configuration file was detected. Adapter module implementers can then use `IOpenConnectRequest::Get<Type>Setting()` methods to update internally used settings.

IResolveLevel2SymbolRequest : Dispatch

This interface is for asynchronously responding to the *OnResolveLevel2Symbol* request. It was introduced with the OpenConnect SDK 5.7.

SendResolvedLevel2Symbol

```
SendResolvedLevel2Symbol( «in» result : ELevel2ReolveResult,
                          «in» resolvedLevel2Symbol : string) : void
```

This sends the result for the *OnResolveLevel2Symbol* request. If the adapter module does not provide a separate symbol for Level2 data, the value of *resolvedLevel2Symbol* should be identical to the name of the security as provided by *OnResolveLevel2Symbol*.

IOnDemandSecurityCatalogue : Dispatch

The CoClass implementing *IOpenConnectAdapterModule* may also implement this interface to enable drill-down (on-demand) security catalogue requests on a folder-by-folder basis.

OnRequestFolder

```
OnRequestFolder( «in» user : string, «in» folderId : string,
                 «in» listRequest : ISecurityListRequest* ) : void
```

Called to obtain the contents of a specific catalogue folder for a given user, or all users if the user parameter is NULL.

OnRequestGroupType

```
OnRequestGroupType( «in» user : string, «in» groupId : string,
                    «in» listRequest : ISecurityListRequest* ) : void
```

Called to obtain the type (folder or list) of a specific catalogue folder for a given user, or all users if the user parameter is NULL. Forwarded when the response for *OnRequestFolder* contains an `<unknown>` node.

IServiceSpec : Dispatch

Used by an Adapter Module to specify the name and capabilities of the service.

SetWindowsServiceName

```
SetWindowsServiceName( «in» windowsServiceName : string ) : void
```

Set the name of the service as it will be registered with the windows Service Control Manager (SCM).

SetFeedServiceName

```
SetFeedServiceName( «in» feedServiceName : string ) : void
```

Set the name of the service as it will be identified by DataConnect (no spaces).

SetSupportsUniqueTickIdentifiers

```
SetSupportsUniqueTickIdentifiers( «in» enabled : bool ) : void
```

Set if the historic tick data will include a unique identifier for each tick to facilitate automatic tick correction.

SetSupportsUserSpecificSecurityAccess

```
SetSupportsUserSpecificSecurityAccess( «in» enabled : bool ) : void
```

Set if access rights must be checked for each user before accessing a security.

SetSupportsUserSpecificSecurityLists

```
SetSupportsUserSpecificSecurityLists( «in» enabled : bool ) : void
```

Set if the symbol lists must be requested for each user or a global list for all users is available.

SetSupportedHistoricDataPeriods

```
SetSupportedHistoricDataPeriods( «in» periodFlags : EDataPeriod ) : void
```

Set which historic data periods are available (flags) from the upstream server.

This is a global setting for all symbols provided by an adapter module. In case the set of periods that are deliverable can change based on the symbol, use EnableFieldCollection from the IMetaFieldSet3 interface (instead of IMetaFieldSet) and set this to:

```
EDataPeriod.PeriodTick | EDataPeriod.Period1Min | EDataPeriod.Period5Min |
EDataPeriod.Period30Min | EDataPeriod.PeriodHourly | EDataPeriod.PeriodDaily |
EDataPeriod.PeriodWeekly | EDataPeriod.PeriodMonthly
```

IConnectionRequest : Dispatch

This interface provides methods to inform about server state change, setting server time and interacting with the log and settings files.

UpdateServerStatus

```
UpdateServerStatus( «in» status : EServerStatus ) : void
```

Update the server status to reflect the current connection state.

UpdateServerTimeStatus

```
UpdateServerTimeStatus( «in» status : ESecurityStatus ) : void
```

Update the time instrument status.

UpdateServerTimeUTC

```
UpdateServerTimeUTC( «in» time : DATE ) : void
```

Update the time in UTC, best called frequently (more than once a minute).

WriteLogMessage

```
WriteLogMessage( «in» logLevel : ELogLevel, «in» message : string ) : void
```

Send a message to the OpenConnect log file. Messages with a log level of Error, Warning or Important will be forwarded to the DataConnect server. By adding certain ini-file settings (see Appendix A) these messages can also be forwarded to connected clients.

GetStringSetting

```
GetStringSetting( «in» section : string, «in» settingName : string,
                  «inout» settingValue : string* ) : void
```

Read a string setting from the OpenConnect configuration file, if the setting is not found in file, the input settingValue will remain unchanged.

GetIntSetting

```
GetIntSetting( «in» section : string, «in» settingName : string,
               «inout» settingValue : int* ) : void
```

Read an integer setting from the OpenConnect configuration file, if the setting is not found in file, the input settingValue will remain unchanged.

GetBoolSetting

```
GetBoolSetting( «in» section : string, «in» settingName : string,
                «inout» settingValue : bool* ) : void
```

Read a Boolean setting from the OpenConnect configuration file, if the setting is not found in file, the input settingValue will remain unchanged.

GetDoubleSetting

```
GetDoubleSetting( «in» section : string, «in» settingName : string,
                  «inout» settingValue : double* ) : void
```

Read a floating point setting from the OpenConnect configuration file, if the setting is not found in file, the input settingValue will remain unchanged.

ISecurityListRequest : Dispatch

This interface is used to send XML Security Lists and Catalogues.

SendListAsXml

```
SendListAsXml( «in» listXml : string ) : void
```

Send the requested XML document.

ISecurityAccessRightsRequest : Dispatch

This interface will receive information about the access rights to a security of a specific user.

SetSecurityAccessRight

```
SetSecurityAccessRight( authorized : bool ) : void
```

Method to define the access rights of a user to a security.

ISubscriptionRequest : Dispatch

This interface is used to set the initial value of a security's fields and obtain an interface to inform about new ticks and corrections.

CreateMetaFieldSet

```
CreateMetaFieldSet( ) : IMetaFieldSet*
```

Factory method to create a meta field set.

SendInitialMetaFields

```
SendInitialMetaFields( «in» fields : IMetaFieldSet*,
                       «out» updateInterface : ISubscriptionUpdates** ) : void
```

Set the initial value of each supported meta field in the security, can also be used to force an update to these fields without introducing new ticks. e.g. name changes.

UpdateSecurityStatus

```
UpdateSecurityStatus( «in» status : ESecurityStatus ) : void
```

Set the status of the security requested.

SecurityNotAvailable

```
SecurityNotAvailable( «in» reason : ESecurityNotAvailable ) : void
```

Mark security as not available.

IMetaFieldSet : Dispatch

This interface is used to set meta data that is needed – but not collected – by DataConnect. If a meta field is not required, the corresponding Set- method should not be called.

Please note: Extended fields and functions are available via the *IMetaFieldSet2* and *IMetaFieldSet3* interfaces.

SetDisplayName

```
SetDisplayName( «in» displayName : string ) : void
```

Set the display name.

SetDisplayNameKanji

```
SetDisplayNameKanji( «in» displayName : string ) : void
```

Set the Kanji characters representing the display name (if applicable).

SetExchangeCode

```
SetExchangeCode( «in» exchangeCode : string ) : void
```

Set exchange code.

SetDisplayUnit

```
SetDisplayUnit( «in» displayUnit : int ) : void
```

Set the display unit, positive values is number of decimal places, negative for the fractional denominator.

SetTradeUnit

```
SetTradeUnit( «in» tradeUnit : string ) : void
```

Set trade unit in Reuters format: Bbl = Barrels, etc.

SetContractSize

```
SetContractSize( «in» contractSize : double ) : void
```

Set the size of the contract in units (defined by tradeUnit field).

SetExpiryDate

```
SetExpiryDate( «in» expiryDate : DATE ) : void
```

Sets the expiry date. Do not call this method for continuous items.

SetTickSize

```
SetTickSize( «in» tickSize : double ) : void
```

Set the minimum possible price movement.

SetCurrencyCode

```
SetCurrencyCode( «in» currencyCode : string ) : void
```

Set the currency code in ISO format: USD, EUR, JPY, GBP, etc.

SetSessions

```
SetSessions( «in» sessions : string ) : void
```

Set the sessions in Reuters format: MO-FR 0800-1700,SU 1900-1400

SetExchangeTimezone

```
SetExchangeTimezone( «in» timezone : string ) : void
```

Set the exchange timezone in Olson format (e.g. Europe/London).

SetDayOpen

```
SetDayOpen( «in» open : double ) : void
```

Set the Daily Open.

SetDayHigh

```
SetDayHigh( «in» high : double ) : void
```

Set the Daily High.

SetDayLow

```
SetDayLow( «in» low : double ) : void
```

Set the Daily Low.

SetDayVolume

```
SetDayVolume( «in» volume : double ) : void
```

Set the Daily Volume.

SetBidN

```
SetBidN( «in» level : int, «in» value : double, «in» size : double ) : void
```

Set the Nth bid level excluding the best bid (i.e. level 1).

SetAskN

```
SetAskN( «in» level : int, «in» value : double, «in» size : double ) : void
```

Set the Nth ask level excluding the best ask (i.e. level 1).

EnableFieldCollection

```
EnableFieldCollection( «in» field : EField ) : void
```

Enable the specified Field for tick collection.

When using OpenConnect 6.2. or above it is advised to use IMetaFieldSet3.EnableFieldCollection for a better control over the creation of a field and the various periods that are supported.

SetTicker

```
SetTicker( «in» ticker : string ) : void
```

Set the ticker information.

SetIsin

```
SetIsin( «in» isin : string ) : void
```

Set the ISIN.

SetSecurityType

```
SetSecurityType( «in» securityType : ESecurityType ) : void
```

Set the security type.

SetBidAndToneN

```
SetBidAndToneN( «in» level : int, «in» value : double, «in» size : double,
                «in» tone : string ) : void
```

Set the Nth bid level excluding the best bid (i.e. level 1) with tone string.

SetAskAndToneN

```
SetAskAndToneN( «in» level : int, «in» value : double, «in» size : double,
                «in» tone : string ) : void
```

Set the Nth ask level excluding the best ask (i.e. level 1) with tone string.

SetDisplayNameForLastField

```
SetDisplayNameForLastField( «in» displayName : string ) : void
```

Set the display name for the last field. The display name must not exceed 20 characters in length and must consist of an alphabetic character followed by alphanumeric or underscore characters. Additionally, the field name cannot be one of the standard fields: VOL, OI, BID, ASK, BVOL, AVOL, ID, IDB, BARTIME. If an invalid value is specified E_INVALIDARG will be returned.

IMetaFieldSet2 : Dispatch

This interface is used to set additional meta data that is needed – but not collected – by DataConnect. If a meta field is not required, the corresponding Set- method should not be called.

Available since OpenConnect version 5.7 in conjunction with DataConect 5.7 and above.

SetRolloverTime

```
SetRolloverTime( «in» hour : int, «in» minute : int ) : void
```

Sets the session's daily rollover time for non-standard sessions that need a rollover other than midnight (hour=0-23, minute=0-59).

SetSourceTimezoneHistoricData

```
SetSourceTimezoneHistoricData( «in» olsonTimezone : string ) : void
```

Sets the timezone of the data provided by history request interface methods in Olson format (e.g. Asia/Tokyo). UTC is assumed when this method is not called.

SetSourceTimezoneDataUpdates

```
SetSourceTimezoneDataUpdates( «in» olsonTimezone: string) : void
```

Sets the timezone of the data provided by subscription updates interface methods in Olson format (e.g. America/New York). UTC is assumed when this method is not called.

SetStringField

```
SetStringField( «in» fieldId : string, «in» fieldValue : string ) : void
```

Generic method to set the string value of a named meta field. See Appendix D – Defining additional fields for more information.

SetIntegerField

```
SetIntegerField( «in» fieldId : string, «in» fieldValue : int ) : void
```

Generic method to set the integer value of a named meta field. See Appendix D – Defining additional fields for more information.

SetDoubleField

```
SetDoubleField( «in» fieldId : string, «in» fieldValue : double ) : void
```

Generic method to set the double value of a named meta field. See Appendix D – Defining additional fields for more information.

SetDisplayNameForUserDefinedField

```
SetDisplayNameForUserDefinedField( «in» userDefinedFieldId : EField, «in»
fieldDisplayName : string ) : void
```

Method to set the display name for one of the user defined fields (`FieldUsrDef1` - `FieldUsrDef10`) that have been optionally enabled for tick collection via `IMetaFieldSet::EnableFieldCollection`.

IMetaFieldSet3 : Dispatch

This interface allows a more fine grained control over the creation of FIDs in DataConnect.

Available since OpenConnect version 6.2 in conjunction with DataConect 6.2 and above.

EnableFieldCollection

```
EnableFieldCollection( «in» field : EField, «in» collect : bool, «in» periods :
EDataPeriod ) : void
```

Enable the specified Field for collection.

The *collect* argument defines, if this field should be automatically enabled and collected when the symbol is newly created in DataConnect (true) or if it should be created on demand as soon as the FID is specifically requested (e.g. by changing the FID value in Tradesignal's instrument properties). It is good practice to delay FID enablement for secondary FIDs to save resources.

The *periods* parameter is a bit set of all the periods that are supported for the respective symbol. For example, if a data feed does only support tick data for a symbol this should be set to `EDataPeriod.PeriodTick`. For a symbol that supports only Daily data and above, this could be set to `EDataPeriod.PeriodDaily|EDataPeriod.PeriodWeekly|EDataPeriod.PeriodMonthly`.

ISubscriptionUpdates : Dispatch

This interface is used to send updates to the collectable fields of a security once it has been initialized and meta fields have been set. All timestamps must be in the timezone specified by the *IMetaFieldSet2.SetSourceTimezoneDataUpdates()* method (UTC by default).

SendNewTick

```
SendNewTick( «in» field : EField, «in» tickId : int64, «in» timestamp : DATE,
             «in» value : double, «in» size : double ) : void
```

Report a new tick (trade, best bid or best ask update, etc.). If a tick already exists with the same timestamp, the new tick will be stored with its timestamp increased by one millisecond.

Please note: Supplying a *tickId* is only supported for the *LAST* field. The *BID* and *ASK* fields will return with *E_INVALIDARG*, if anything but the *UnknownTickId* is supplied.

SendTickUpdate

```
SendTickUpdate( «in» field : EField, «in» tickId : int64, «in» timestamp : DATE,
                «in» value : double, «in» size : double ) : void
```

Update the value or volume of a historic tick identified by the *tickId*. The configuration file setting *IndexTradeIds* can be used to improve performance of this method.

Important: It is not recommended to use this function to insert new ticks, the database search to verify if the tick exists can be time consuming, the *SendNewTick()* function is optimized for inserting new ticks with no overhead. If, despite this recommendation, ticks are inserted with this method, be aware that if a tick already exists with the same timestamp, the new tick will be stored with its timestamp increased by one millisecond.

SendDeleteTick

```
SendDeleteTick( «in» field : EField, «in» tickId : int64 ) : void
```

Remove a historic tick identified by the *tickId*.

IBarHistoryRequest : Dispatch

This interface is used to send chunks of cumulated historical data. All timestamps must be in the timezone specified by the *IMetaFieldSet2.SetSourceTimezoneHistoricData()* method (UTC by default).

CreateHistoricChunk

```
CreateHistoricChunk( ) : IBarChunk*
```

Factory method to create a data chunk container.

SendHistoricChunk

```
SendHistoricChunk( «in» dataChunk : IBarChunk*, «in» moreToFollow : bool ) : bool
```

Send a chunk of data in sequence and inform if more data will be sent. Data is sent from newest to oldest. If *sendMoreChunks* is set to false when the function returns, further data must not be sent.

Please note: If two or more bars with the same timestamp are sent, these will update previously inserted bars instead of inserting duplicate data.

IBarChunk : Dispatch

This interface is used to prepare a chunk of cumulated data to send to the OpenConnect service. All timestamps must be in the timezone specified by the *IMetaFieldSet2.SetSourceTimezoneHistoricData()* method (UTC by default).

Very important: Data must always be organized newest to oldest.

Add

```
Add( «in» timestamp : DATE, «in» open : double, «in» high : double,
     «in» low : double, «in» close : double, «in» size : double ) : void
```

Add a bar to the data set.

ITickHistoryRequest : Dispatch

This interface is used to send chunks of historical tick data. All timestamps must be in the timezone specified by the *IMetaFieldSet2.SetSourceTimezoneHistoricData()* method (UTC by default).

CreateHistoricChunk

```
CreateHistoricChunk( ) : ITickChunk*
```

Factory method to create a data chunk container.

SendHistoricChunk

```
SendHistoricChunk( «in» dataChunk : ITickChunk*, «in» moreToFollow : bool ) : bool
```

Send a chunk of data in sequence and inform if more data will be sent. Data is sent from newest to oldest. If *sendMoreChunks* is set to false when the function returns further data must not be sent.

Please note: If two or more trades with the same timestamp are sent, these will update previously inserted trades instead of inserting duplicate data.

ITickChunk : Dispatch

This interface is used to prepare a chunk of tick data to send to the OpenConnect service. All timestamps must be in the timezone specified by the *IMetaFieldSet2.SetSourceTimezoneHistoricData()* method (UTC by default).

Very important: Data must always be organized newest to oldest.

Add

```
Add( «in» tickId : int64, «in» timestamp : DATE, «in» value : double,
     «in» size : double ) : void
```

Add a tick to the data set.

ITickConfirmationRequest : Dispatch

This interface is used to provide a valid tick id for user created data. Timestamps are in the timezone specified by the *IMetaFieldSet.SetExchangeTimezone()* method.

ConfirmNewTick

```
ConfirmNewTick( «in» timestamp : DATE, «in» tickId : int64, «in» value : double,
                «in» size : double ) : void
```

Method to confirm a user created tick with a new tick ID.

Constants : Module

All constants values used by the library.

UnknownTickId An unknown ID for a tick.

EDataPeriod : Enum

The various periods used to represent historical data.

PeriodTick Each individual trade or price change as a data point.

Period1Min Data points representing one minute of activity, timestamp is the end of the inclusive period, 12:00 = 11:59:00:001 – 12:00:00:000.

Period5Min Data points representing five minutes of activity aligned to hour boundaries, timestamp is the end of the inclusive period, 12:10 = 12:05:00:001 – 12:10:00:000.

Period30Min Data points representing 30 minutes of activity aligned to hour boundaries, timestamp is the end of the inclusive period, 12:30 = 12:00:00:001 – 12:30:00:000.

PeriodHourly Data points representing one hour of activity aligned to hour boundaries, timestamp is the end of the inclusive period, 10:00 = 09:00:00:001 – 10:00:00:000.

PeriodDaily Data points representing one day of activity, timestamp is the start of the day, 01/01/2009 00:00:00 = 00:00:00:000 – 23:59:59:999.

PeriodWeekly Data points representing one week of activity, timestamp is the start of the Sunday that ends the week.

PeriodMonthly Data points representing one month of activity, timestamp is the start of the last day of the inclusive month.

ESecurityStatus : Enum

The various states an instrument (security or time) can be in.

StatusActive Updates are normally processed.

StatusStale Updates are temporarily delayed or a (recoverable) problem with the upstream data feed connection prevents updating the symbol.

StatusClosed No updates will be sent anymore. This should be used for unrecoverable errors only (e.g. a realtime contract that was previously generating updates and that has just expired or a symbol that has been ultimately dropped off the upstream source). No more realtime updates are expected after receiving StatusClosed for a symbol.

EServerStatus : Enum

The various states a server can be in.

ServerConnecting	When transitioning from disconnected to connected.
ServerConnected	Normal connected state.
ServerDisconnecting	When transitioning from connected to disconnected as a result of a MANUAL stop.
ServerDisconnected	When manually shut down.
ServerReconnecting	When connection is dropped not as a result of a manual action and automatic reconnection is in progress.
ServerStale	When connected but data is delayed or stalled.

ELogLevel : Enum

The various levels of log messages used to indicate log message importance and destination.

LogInfo	Used to log general messages to the log file only.
LogImportant	Used to log significant (e.g. start/stop) messages to the log file and windows log.
LogWarning	Used to log warning messages to the log file and windows log.
LogError	Used to log error messages to the log file and windows log.
LogDebug	Used to log messages to the log file, only if the '[Log]/LogDebug' setting is set to 1.

EField : Enum

Enumeration of the various fields that are supported for historic tick collection.

FieldLast	Last trade.
FieldBestBid	Best bids.
FieldBestAsk	Best asks.
FieldOpenInterest	Open interest.
FieldUsrDef1 … FieldUsrDef20	User defined fields (there are actually up to 100 user defined fields available, to access fields beyond 20, use FieldUsrDef1 + *offset* where 20 <= *offset* <= 99). The content depends on the adapter module's implementation. These fields can be named via the IMetaFieldSet2 interface.

ESecurityNotAvailable : Enum

The possible reasons why a security might not be available.

SecurityNotExisting	Used to indicate that a subscription request failed because it does not exist.
SecurityNotPermissioned	Used to indicate that a subscription request failed due to a lack of permissions.

ECorrectionType : Enum

The various manual tick corrections that can be sent to the upstream server.

CorrectionUpdate	An update to the value or volume of an existing tick.
CorrectionDelete	An existing tick has been deleted.
CorrectionNew	A new tick has been added.

ESecurityType : Enum

The type of a security.

SecurityTypeUndefined

SecurityTypeIndex

SecurityTypeStock

SecurityTypeFuture

SecurityTypeRollingFuture

SecurityTypeCurrency

SecurityTypeOption

SecurityTypeWarrant

SecurityTypeCertificate

SecurityTypeBond

SecurityTypeOther

SecurityTypeFutureSpread	Added with OpenConnect SDK 5.7
SecurityTypeGenericSpread	Added with OpenConnect SDK 5.7

SecurityTypeGenericSpot	Added with OpenConnect SDK 5.7
SecurityTypeGenericFixing	Added with OpenConnect SDK 5.7

ELevel2ResolveResult : Enum

Success and error codes for resolving a Level2 symbol introduced with the OpenConnect SDK 5.7.

Success	The Level2 symbol has been successfully resolved.
NotAvailable	There is no Level2 available for the requested symbol.
NoPermission	Missing entitlement to access Level2 data for a given symbol.
UnspecifiedError	The Level2 symbol could not be resolved due to an unspecified error.

Appendix A – OpenConnect.ini Settings

Section General

AdapterModuleClassId

Set the GUID of the COM class in the Adapter Module that implements the *IOpenConnectAdapterModule* interface. This value must be in registry format (e.g. {6C4D8623-C999-4180-A0F6-F986F610165B}). Unless this value is set OpenConnect will not start.

PushPort

Set the TCP/IP port that the service will use for clients to obtain data updates. Default value is 27367.

BackfillPort

Set the TCP/IP port that the service will use for clients to obtain historic and entitlement data. Default value is 27366.

PersistentSymbols

Set this value to 0 to instruct DataConnect that the Adapter Module is managing the storage of all data and therefore not to store the data a second time. Disabling this mode will mean that the requests for historic data may be more frequent as users open and close charts. Default value is 1 (on).

Important: Only disable the persistent storage of symbols if the Adapter Module provides data in all supported periods (from tick up to monthly). If only tick data is provided, for example, the requirement to recumulate daily data for each history request (which may occur many times per day in this mode) can incur a substantial performance penalty.

IndexTradeIds

Set this to 1 to instruct the DataConnect service to index the unique identifiers associated with a specific trade, this will substantially boost performance of the *ISubscriptionUpdates.SendTickUpdate()* method (and incur a small to moderate degradation to tick insert performance). This option should be turned on where tick corrections are commonplace, and there are not too many new ticks per minute per instrument. Default value is 0 (off).

Please note: Changes to this value will only be applied to newly created instruments.

FieldDefinitionFile

This specifies the name of the file containing definitions for additional instance field. It defaults to “FieldDefinitions.xml”. See Appendix D – Defining additional fields for more information.

SupportsCurrencyConversion

OpenConnect feeds can be used for currency conversion in a master/slave configuration when set to 1. The default is 0 (turned off). See Appendix E – Supplying Currency Conversion for more information.

Section Logging

LogFile
The name and path of the log file. Default value is ‘OpenConnect.log’.

LogMaxSize
The maximum size of the log file in bytes. If the log file reaches this size a backup of the current log file will be created and a new log file started. The backup log file will have ‘.old’ appended to its filename. Default value is 16777216 (16MB).

LogDebug
Set to 1 to enable (0 to disable) logging of messages marked with the severity LogDebug. Default value is 0.

LogRecordCreation
Set to 1 to enable (0 to disable) logging of subscription to symbols. Default value is 0.

LogRecordClients
Set to 1 to enable (0 to disable) logging of individual client’s interest in active symbol subscriptions. Default value is 0.

LogRecordCache
Set to 1 to enable (0 to disable) logging of changes to the internal subscription cache. Default value is 0.

LogRecordUpdates
Set to 1 to enable (0 to disable) logging of updates (ticks, corrections, deletions) passing through an active symbol subscription. Default value is 0.

LogBackfill
Set to 1 to enable (0 to disable) logging of backfill information. Default value is 0.

LogTime
Set to 1 to enable (0 to disable) logging of information about service time processing and publishing. Default value is 0.

LogTMDSRequests
Set to 1 to enable (0 to disable) logging of information on incoming requests from the DataConnect service. Default value is 0.

LogEventLoop
Set to 1 to enable (0 to disable) logging of information on how much time threads spend in the message loop. Default value is 0.

LogErrorForwardToClient
Set this to 1 if all log messages having severity "Error" should also be forwarded to all connected Tradesignal clients to be displayed as an alert. The default is 0. (Added with the OpenConnect SDK 5.7)

LogWarningForwardToClient

Set this to 1 if all log messages having severity "Warning" should also be forwarded to all connected clients to be displayed as an alert. The default is 0. (Added with the OpenConnect SDK 5.7)

LogImportantForwardToClient

Set this to 1 if all log messages having severity "Important" should also be forwarded to all connected clients to be displayed as an alert. The default is 0. (Added with the OpenConnect SDK 5.7)

Section Internal

TimeSymbol

Specify the name of the symbol that will be advertised to DataConnect as providing time information. This value must match the symbol entered into the MDS.ini file. This value should very rarely, if ever, be changed. Default value is '.EVAS'.

CacheSeconds

Define the time in seconds a symbol remains active, when the last client dropped the connection to a subscription record. Default value is 10.

MaxMsInEventLoop

Define the maximum time in milliseconds threads spend in the message loop before they will be forced to go back to their normal operation. Default value is 5000.

Appendix B – Typical function call scenarios

The following diagram shows the typical sequence of calls at various times during the lifetime of an Adapter Module.

DataConnect Service Start Up

Open Connect Server	Adapter Module	
OnRegisterModule		
OnConnect		
	UpdateServerStatus	
	UpdateServerTimeStatus	
	UpdateServerTimeUTC	
OnSubscribeSecurity		
	SecurityNotAvailable	*If requested security is not existing*
	SendInitialMetaFields	*If requested security exists*
	UpdateSecurityStatus	
	SendNewTick	*Trade Updates*
OnRequestHistory		
	SendHistoryChunk	*Requests any missing data*

DataConnect Service Shutdown

Open Connect Server	Adapter Module
OnUnsubscribeSecurity	
OnDisconnect	
	UpdateServerStatus

Security added

Open Connect Server	Adapter Module	
OnSubscribeSecurity		
	SecurityNotAvailable	*If requested security is not existing*
	SendInitialMetaFields	*If requested security exists*
	UpdateSecurityStatus	
	SendNewTick	*Trade Updates*
OnRequestHistory		
	SendHistoricChunk	

Security deleted

Open Connect Server	Adapter Module
OnUnsubscribeSecurity	
	UpdateSecurityStatus

Normal operation

Open Connect Server	Adapter Module	
	UpdateServerStatus	*When problems occur*
	UpdateServerTimeUtc	
	SendNewTick	
	SendTickUpdate	*Server tick correction*
	SendDeleteTick	*Server tick correction*
	UpdateSecurityStatus	*Stale, etc.*
OnTickUpdated		*User manually corrects a tick*

New user logs on and requests securities

Open Connect Server	Adapter Module
OnRequestSecurityListCatalogue	
	SendListAsXml
OnRequestSecurityList	
	SendListAsXml
OnRequestSecurityAccessRights	
	SetSecurityAccessRight

Appendix C – Passing time stamps in interface methods when using C++

Time stamps are used in various situations, for example when asking for tick history starting at a given date (`IConnectAdapterModule::OnRequestTickHistoryFromDate`) or when providing the data via `ITickChunk::Add`. OpenConnect supports stamps with millisecond granularity.

Time stamps are passed in COM interfaces via the automation data type called `DATE` (which basically is a `typedef` of a double precision floating point, with sufficient precision to transport time stamps with millisecond granularity). When the adapter module is implemented using the .NET runtime (e.g. when using the C# language), you can skip the remainder of this section as the .NET `DateTime` class supports millisecond granularity and will convert correctly when passed as a `DATE`.

When implementing the adapter module using unmanaged C++, there are a few caveats to consider when converting to or from the `DATE` data type. The usual way to convert to a `DATE` is to use Win32 functions like `SystemTimeToVariantTime` or `VariantTimeToSystemTime` or to use the `COleDateTime` class (which utilizes these functions under the hood). The problem is that these functions and the `COleDateTime` helper class have a limited granularity of only up to one second. Conversion to or from a `DATE` will thus lose milliseconds precision. The code below demonstrates how to avoid this problem. Note that in the following examples we are utilizing `SYSTEMTIME` as a milliseconds aware date/time structure; `TIMESTAMP_STRUCT` (when using ODBC) or `FILETIME` could also be used.

Converting from SYSTEMTIME to DATE keeping millisecond precision

```
// the following example shows how to create a DATE with milliseconds granularity:
SYSTEMTIME timestampWithMs = ...; // assume timestampWithMs gets populated by data source

// the following call to SystemTimeToVariantTime will lose the millisecond granularity
// we are not passing in the millisecond part because this may lead to rounding.
int savedMilliseconds = timestampWithMs.wMilliseconds;
timestampWithMs.wMilliseconds = 0;
DATE dateTimeWithoutMs;
if (!SystemTimeToVariantTime(&timestampWithMs, &dateTimeWithoutMs))
    ... // handle error

// add milliseconds saved above
const int millisecondsPerDay = 1000 * 60 * 60 * 24;
DATE dateTimeWithMs = dateTimeWithoutMs + (1.0 / millisecondsPerDay) * savedMilliseconds;

// example usage: populate tick chunk
CComPtr<ITickChunk> pChunk;
pTickHistoryRequest->CreateHistoricChunk(&pChunk);
pChunk->Add(dateTimeWithMs, open, high, low, close, size);
```

Converting from DATE to SYSTEMTIME keeping millisecond precision

```
// example usage: cracking a DATE passed in via OnRequestTickHistoryFromDate()
DATE dateTimeWithMs;
OnRequestTickHistoryFromDate(security, field, pHistoryRequest, dateTimeWithMs);

// calculate millisecond part from given DATE
const int msPerSecond = 1000;
const int msPerDay = msPerSecond * 60 * 60 * 24;
int savedMs = (((int)((fmod(dateTimeWithMs, 1.0) * msPerDay) + .5)) % msPerSecond);

// like SystemTimeToVariantTime, VariantTimeToSystemTime may involve rounding
// we avoid this by not passing in the milliseconds
SYSTEMTIME timestampWithMs;
if (!VariantTimeToSystemTime(dateTimeWithMs - savedMs * (1.0/msPerDay), &timestampWithMs))
    ... // handle error

// set milliseconds in SYSTEMTIME structure
timestampWithMs.wMilliseconds = savedMs;
```

Appendix D – Defining additional fields

Tradesignal allows displaying symbol-related, additional information delivered by the upstream data provider in form of so-called quote fields. Fields, which may be textual or numeric information, can be displayed as columns in watchlist, scanner, or portfolio documents. Examples for quote fields would be the last price, the date of the last trade, today's high price, or the currency a symbol is traded in.

The `IMetaFieldSet` interface already allows defining a set of predefined, common fields that can also be accessed in the way described above. Starting from version 5.5, OpenConnect and DataConnect allow user-defined, arbitrary quote fields, defined through a dedicated XML document and set through `IMetaFieldSet2` (`SetStringField, SetIntegerField, SetDoubleField`).

A field definition contains of a unique integer id, a short name (acronym), a short display name (which will be used by Tradesignal to label columns), an optional explanatory description and a data type indication. See the following XML schema for a description of the XML definition file:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="fieldId">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="500000"/>
      <xs:maxInclusive value="599999"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="fieldAcronym">
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z][a-zA-Z0-9_]*"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="fieldDisplayName">
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="fieldType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="int"/>
      <xs:enumeration value="double"/>
      <xs:enumeration value="string"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="field-definitions">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="field" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="id" type="fieldId" use="required"/>
            <xs:attribute name="name" type="fieldAcronym" use="required"/>
            <xs:attribute name="display" type="fieldDisplayName" use="required"/>
            <xs:attribute name="type" type="fieldType" default="string"/>
            <xs:attribute name="description" type="xs:string" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The following represents an example field definition xml that describes a floating point, an integer, and a text field:

```
<?xml version="1.0" encoding="utf-8"?>
<field-definitions>
  <field id="500000" name="STRIKE_RATIO" display="Strike Ratio" type="double"
       description="The number of equity shares per warrant."/>
  <field id="500001" name="TODAY_BID_NUM" display="Bids Today" type="int"
       description="The number of bids made for an instrument today."/>
  <field id="500002" name="MKT_M_NAME" display="Market Maker Name" type="string"
       description="The name of the market maker."/>
</field-definitions>
```

The field definition file will be read when the OpenConnect service starts up. The default location of the file is `FieldDefinitions.xml` in the same directory where the `OpenConnect.ini` resides. If there is the need to relocate and/or rename the field definition file, this can be achieved by pointing the setting `[General] FieldDefinitionFile` to the new location.
When field definitions have to be added or changed while the OpenConnect service keeps on running, this can be achieved by instructing DataConnect to reload the field definitions using the `reloadfields` command, e.g. `reloadfields CSV:` would reload the field definitions from the DataConnect extension service having the prefix `CSV:`. DataConnect will also notify connected Tradesignal clients of the updated fields so that new fields can be immediately made available in Tradesignal.

The following sample source code will exemplify how to update the values for the three fields defined above and then how to instruct forwarding them to DataConnect (and on to connected Tradesignal clients):

```
// create meta field set via request previously retrieved
IMetaFieldSet metaData = request.CreateMetaFieldSet();
IMetaFieldSet2 metaData2 = (IMetaFieldSet2)metaData;

// set/update user-defined fields
metaData2.SetDoubleField("STRIKE_RATIO", 123.45);
metaData2.SetIntegerField("TODAY_BID_NUM", 27367);
metaData2.SetStringField("MKT_M_NAME", "XYZ Bank")

// forward update to clients
ISubscriptionUpdates updateInterface;
request.SendInitialMetaFields(metaData, out updateInterface);
```

Appendix E – Supplying Currency Conversion Information

Tradesignal and DataConnect allow converting the currency a security is quoted in to a different target currency. Security prices are then historically adapted using rates from the base and the target currency spanning the same time frame as the underlying instrument.

OpenConnect related settings

Introduced in version 5.5, OpenConnect adapter modules can provide currencies for conversion purposes. The following OpenConnect.ini setting must be set to enable this feature (restart necessary):

```
[General]
SupportsCurrencyConversion=1
```

Information on the available set of currencies has to be exposed in a dedicated XML definition file adhering to the following XML schema:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="currencies">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="currency" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="symbol" type="xs:string" />
            <xs:attribute name="base" type="xs:string" use="required"/>
            <xs:attribute name="inverted" type="xs:boolean" default="0"/>
            <xs:attribute name="factor" type="xs:nonNegativeInteger" default="1"/>
            <xs:attribute name="name" type="xs:string" use="required" />
            <xs:attribute name="iso" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The default file name used for this purpose is CurrencyDefintions.xml in the same directory where OpenConnect.ini resides. If necessary, the following setting is available to specify an alternative file path:

```
[General]
CurrencyDefinitionFile=...
```

Here is the content of an example currency definition file:

```
<?xml version="1.0" encoding="utf-8"?>
<currencies>
  <currency iso="AUD" base="USD" factor="1"   inverted="1" name="AUSTRALIAN DOLLAR"    symbol="AUDUSD" />
  <currency iso="AUc" base="AUD" factor="100" inverted="1" name="AUSTRALIAN CENTS"     symbol="" />
  <currency iso="CAD" base="USD" factor="1"   inverted="0" name="CANADIAN DOLLAR"      symbol="USDCAD" />
  <currency iso="CAc" base="CAD" factor="100" inverted="0" name="CANADIAN CENTS"       symbol="" />
  <currency iso="CHF" base="USD" factor="1"   inverted="0" name="SWISS FRANC"          symbol="USDCHF" />
  <currency iso="EUR" base="USD" factor="1"   inverted="1" name="EURO"                 symbol="EURUSD" />
  <currency iso="EUc" base="EUR" factor="100" inverted="1" name="EURO CENTS"           symbol="" />
  <currency iso="GBP" base="USD" factor="1"   inverted="1" name="BRITISH POUND"        symbol="GBPUSD" />
  <currency iso="GBp" base="GBP" factor="100" inverted="1" name="BRITISH PENCE"        symbol="" />
  <currency iso="USD" base="USD" factor="1"   inverted="1" name="UNITED STATES DOLLAR" symbol="" />
  <currency iso="USc" base="USD" factor="100" inverted="1" name="UNITED STATES CENTS"  symbol="" />
</currencies>
```

The entries present in this XML file will constitute the list of target currencies displayed in the Currency field of Tradesignal's instrument property inspector.

The “iso” attribute has to contain a target currency referring to an ISO 4217 three letter code. The “base” attribute defines the reference currency, which is usually USD. The exception to this rule are subdivision currencies like the Euro cent (EUc), which is worth 1/100th of one unit of its base (EUR). The “factor” attribute for EUc is therefore 100.

The “symbol” attribute defines the OpenConnect adapter’s symbol code to be used for this currency.

The “inverted” flag defines how the currency is quoted – direct (inverted=”0”) or indirect (inverted=”1”) to the base, e.g. target against base like in symbol AUDUSD above or base against target like in USDCAD. AUDUSD: Amount of AUD per USD; USDCAD: Amount of USD per unit of CAD.

Note that changing the currency definition file demands restarting the OpenConnect service.

DataConnect related settings

DataConnect forwards currency information to Tradesignal that is provided by one particular data feed. In case more than one data feed is configured, the DataConnect Console can specify, which data feed to ask for currency data when a conversion needs to be calculated. This can be done via Settings > Data Service (General) > Data feed used to load data for currency conversion.

When the provided currency conversion file is changed, template pages should be reloaded to keep DataConnect and Tradesignal up to date with all available conversion currencies. This can be achieved in the DataConnect Console via Application Button > Run Nightly Maintenance > Force Template Reload. Note that currently connected Tradesignal clients may have to reconnect (or alternatively restart) in order to use updated currency information.

Appendix F – Supporting non-default Weekly Candles

Usually weekly candles span from Monday (first inclusive day) till Sunday (last inclusive day). However, this is not true for every country, e.g. some Middle Eastern places have a different notion of weekends, and thus, what is generally considered as the first day of the week will be different from the Western view.

In Dubai, for example, the two weekend days are Friday and Saturday and thus the first day of the week falls on the Sunday. In Saudi Arabia this looks a bit different again, i.e. in Riyadh the weekend is celebrated on Thursdays and Fridays, so the first day of the working week will fall on a Saturday.

DataConnect allows OpenConnect adapter module implementers to define the first day of the week on a per symbol basis through the `IMetaField2` interface and the `SetStringField` function, providing *"`OC_WEEK_WORKDAY1`"* as the field identifier and an English week day to be used as the first day of the week (*"`Monday`"*, *"`Tuesday`"*, ..., or the respective three letter abbreviations *"`MON`"*, *"`TUE`"*, ...). If such a field is not explicitly populated, the default value (the Monday) will apply.

Note that if you want to change a symbol's weekly candle definition by providing an update for this field, you will have to instruct DataConnect to re-summarize existing data for affected instruments afterwards (e.g. by forcing a session template update through the nightly maintenance, available via the Console's default menu).