FACTSET 〉 SEE THE ADVANTAGE

FactSet DataFeed API

COM Programmer's Manual and Reference
Version 3.0 A

Table of Contents

Notice

FactSet Consulting Services

North America - FactSet Research Systems Inc.	
United States and Canada	+1.877.FACTSET
Europe – FactSet Limited	
United Kingdom	0800.169.5954
Belgium	800.94108
France	0800.484.414
Germany	0800.200.0320
Ireland, Republic of	1800.409.937
Italy	800.510.858
Netherlands	0800.228.8024
Norway	800.30365
Spain	900.811.921
Sweden	0200.110.263
Switzerland	0800.881.720
European and Middle Eastern countries not listed above	+44.(0)20.7374.4445
Pacific Rim- FactSet Pacific Inc.	
Japan Consulting Services (Japan and Korea)	0120.779.465 (Within Japan) +81.3.6268.5200 (Outside Japan)
Hong Kong Consulting (Hong Kong, China, India, Malaysia, Singapore, Sri Lanka, and Taiwan)	+852.2251.1833
Sydney Consulting Services	1800.33.28.33 (Within Australia) +61.2.8223.0400 (Outside Australia)
E-mail Support	
	support@factset.com

Document Organization and Audience

This document is intended for application programmers that are familiar with Microsoft Visual Studio and Object-Oriented Systems. Its purpose is to fully describe the functionality contained within the FactSet DataFeed COM API. This document is intended to be read cover-to-cover, and then act as a reference guide to application developers using the FactSet DataFeed COM API.

- Chapter 1 - Introduces FactSet DataFeed COM API and defines key concepts and terminology.
- Chapter 2 - Explains how to build and link applications using this API.
- Chapter 3 - Describes the programming concepts at various stages of an application.
- Chapter 4 - Lists the complete Class Reference.
- Chapter 5 - Describes the Permissioning Service
- Appendix - Extends the class reference by provided additional details.

Document Convention

This document uses the following conventions:

- Code snippets use a courier 10 font - `FDF::connect()`
- Methods, when first introduced, appear in bold - **`FDF::dispatch()`**
- The directory delimiter character follows the Windows convention - backslash ('\')
- Items of importance will be in boxes of following type:

❖ ***Important notations will be in this type of box.***

Trademarks

FactSet is a registered trademark of FactSet Research Systems, Inc.

Microsoft is a registered trademark, and Windows is a trademark of Microsoft Corporation.

All other brand or product names may be trademarks of their respective companies.

Acknowledgements

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org).

Chapter 1 Introduction

1.1 The FactSet DataFeed COM API

The FactSet DataFeed COM API is a multi-platform COM object-oriented framework which is used to communicate with a FactSet data source. The API assists developers with all aspects of communication, request/message processing, and subscription management. The classes simplify data access by providing asynchronous messages to application-defined callbacks.

Applications have two choices when connecting to a data source: a FactSet Data Server or the local FactSet workstation. The chosen data source will authenticate as well as permission the various data sets available. Applications that attempt to connect without authorization will receive a connection error. Connected applications that request data they are not entitled to receive will receive an error message from the data source.

The first data source option is the FactSet Data Server, which is a back end system that is hosted by FactSet. Connections to a FactSet Data Server occur over the Internet or a WAN via TCP/IP. Applications must be given a username, password, and the address information (i.e., IP and port number) for the FactSet Data Server.

The second data source option is a local FactSet workstation, which uses the user's existing FactSet terminal installation along with the permissions tied to that user's serial number. Connections to a local FactSet workstation occur on the user's local machine via COM and TCP/IP. Applications must be given a username and serial number. This configuration is designed for the consuming application to receive data just for local use on the workstation, not for sharing data to any other user.

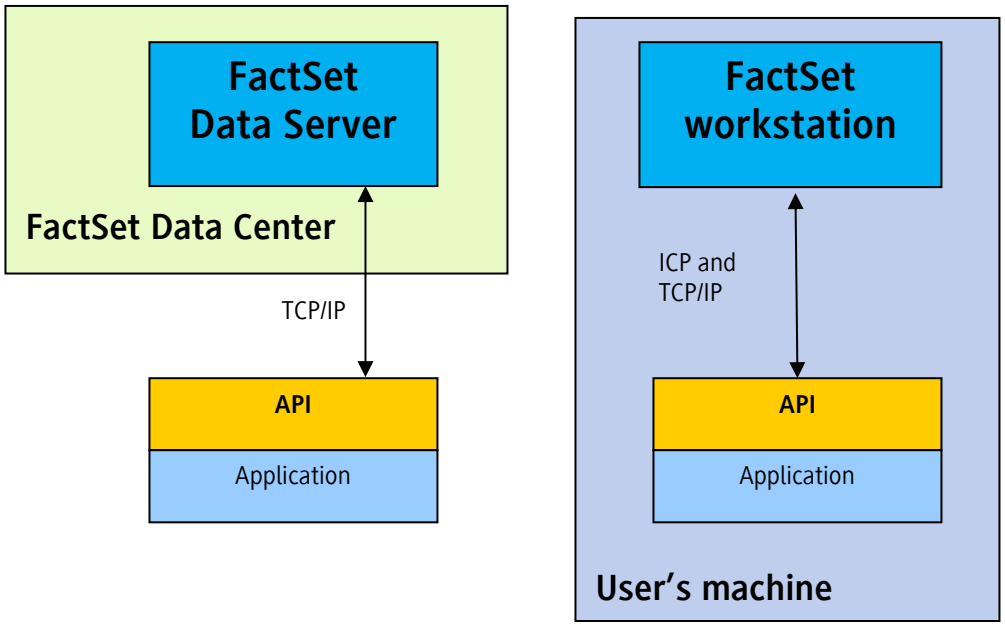


Figure 1: Two API clients connected to the two different FactSet data sources

1.2 Terminology

The following terminology is used throughout this documentation:

Terminology	Meaning
API	Application Programming Interface - a set of defined interfaces that applications use to extract information from the FactSet Data Server.
COM	Component Object Model – a specification and a set of services that allow for the creation of object-oriented, distributed applications using a number of programming languages
SDK	Software Development Kit - a collection of libraries, include files, documentation, and sample codes that make up this toolkit.
XML	eXtensible Markup Language - a defined standard for exchanging information. The information contains markup tags used to describe the data values.
TCP/IP	Transport Control Protocol over Internet Protocol - the protocol that this API uses to communicate to the FactSet Data Server.
FactSet Data Server	A server which provides permissioned access to FactSet data.
FDS	Multiple meanings. FDS is the ticker symbol for FactSet Research Systems Inc. It may also stand for the FactSet Data Server. The meaning is defined by its context.
Service	A data source or supplier identified by a string name.
FDS1	FactSet Data Service Version 1 - a well-known data service. For a complete description of the data fields, types, and possible values see the *FactSet Data Service Specification*.
FDS_FUND	FactSet's Fundamental Data Service. Used for End of Day data.
FDS_C	FactSet's Canned Data Service. Recorded data is replayed, used for testing.
FDS_PERM	FactSet's Permission Service. Used by third party integrators to enforce end-users Exchange permissions using the Workstation Entitled API setup.
Consumer	Any application that uses this API.
Stream	A virtual tunnel of messages for a given request.
Tag	An integer resource used to identify a particular stream.
FDF	A singleton class used to communicate with the FactSet Data Server. It is the abbreviation for FactSet Data Feed.
Event Handler	An application-defined method that is invoked by the API.
FID	Field Identifier - an integer identifier that describes the encoding and business meaning of a field value.
Opaque Data	Data without a defined interpretation, which is simply a pointer and size to the data.
Field/Value Pairs	A self-describing message format used in API messages. Each pair contains a FID and some opaque data. The FID defines the type and meaning of the data.

1.3 High Level Overview

The following diagram shows the logical connections to the FactSet Data Server:

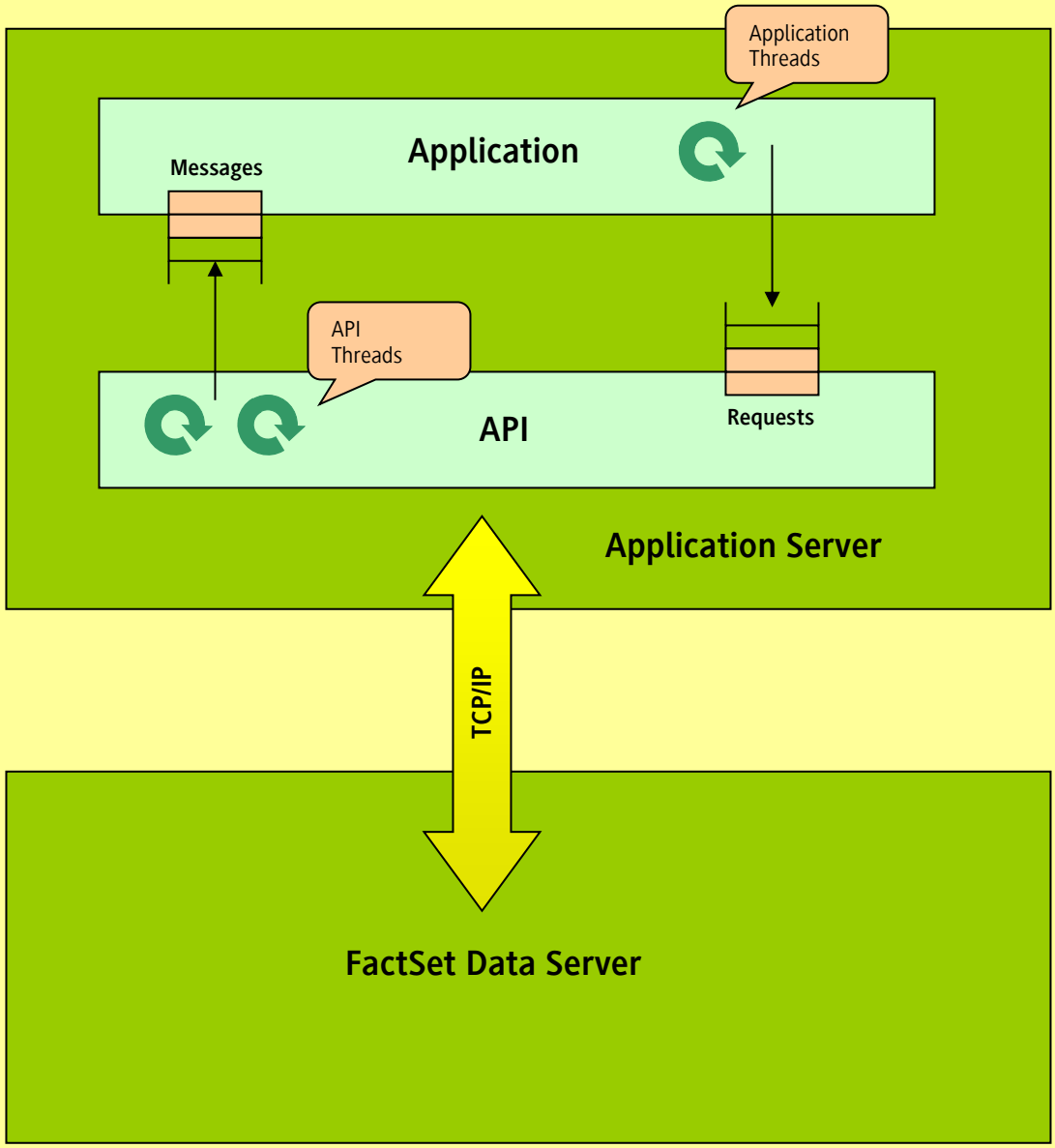


Figure 2: High Level Overview

Applications will use the interface defined by the API to do the following:

- **Connect to the Data Server**: This will initiate the TCP connection and start an internal communication thread within the API.
- **Request Data**: Requests will be posted on a queue to be sent out via the communication thread.
- **Receive Messages via Event Handlers**: Messages will be posted to a message queue by the communication thread. The dispatch window will fire a message event for each message. All events will be handled in the context of an application thread.
- **Disconnect from the Data Server**: The application may disconnect from the Data Server at any time. This will destroy the communication thread as well.

Information on API Threads

The API will create one thread per FDF object. This thread serves as a communication thread and is responsible for all of the TCP/IP communication with the data server. The thread is created when the application connects to the FactSet Data Server, and destroyed when the application calls disconnect().

In addition, the API starts a single global maintenance thread. This thread will be created only if the application uses the Logging interfaces within the API. Once this thread is created, it can only be destroyed on program termination.

1.4 API Core Functionality and Benefits

The API provides the following services to applications:

- Support for multiple development platforms
- Abstract the underlying TCP/IP connection
- TCP connection failure handling
- Simplified data access
- A consistent interface for opening and closing streams
- Subscription Management
- Logging and Configuration Management
- Class-thread-safe, thread-aware

1.4.1 Support for Multiple Development Platforms

Multiple development environments are supported by the API. Any language that supports COM objects can be used to develop applications with this toolkit, such as Visual Basic (6 and .NET), C#, C++ and VBScript.

FactSet also has native C++ and Java API toolkits as well.

1.4.2 TCP/IP Communication

The API handles all aspects of the TCP/IP connection to the Data Server including problems related to asynchronous communication, byte-ordering, and the buffering needed when using stream-oriented protocols.

The API will detect TCP network failures, and will notify all open streams of the condition (i.e., each stream will receive a stale message). Applications only need to monitor the individual streams, and not the connection as a whole.[1]

The API will continuously retry the connection to the Data Server in the event of a TCP disconnect. Upon a successful reconnect, the current open streams will also be re-established. Refresh data will be sent and each open stream will transition from a stale to a non-stale state.

Required Ports:

- tcp/6681 – Connection to Exchange DataFeed Server
- tcp/443 – Web-based authentication

api(-stage).df.factset.com and canned-stage.df.factset.com: tcp/6681 need to be opened outbound-initiated for subnets:

- 192.234.235.0 (255.255.255.0)
- 64.209.89.0 (255.255.255.0)

1.4.3 Security Protocols

Clients should not hardcode dependencies on any specific security protocol as FactSet is continuously reviewing security policies and reserves the right to disable support for older security protocols with short notice[2]. The current supported protocols are TLSv1.1 and TLSv1.2 but at a future date, these may be replaced with future versions. Clients should make sure that their software can handle ever changing Security Protocols.

1.4.4 Simplified Data Access

The API delivers data using field/value pairs. The RT_Message class allows applications to easily extract the data fields. This class supports both random and sequential access.

1.4.5 Request Consistency

The API provides a consistent interface for opening and closing streams. All requests will receive an integer identifier (tag) to a virtual stream. This applies to both static (i.e., snapshot) and dynamic requests. In addition, messages on any given stream will be associated by the stream identifier. To close a stream, the application needs to pass the stream id back to the API.

---

[1] The API does inform the application about the connection status as a whole, and the application can use this information in any way it sees fit.
[2] As of 29-Jul-2017 support for security protocol TLSv1.0 is disabled and requests using this TLS version will fail.

1.4.6 Subscription Management

The API allows applications to request duplicate data items. Each item will create its own stream. Although the virtual streams are independent they will receive identical messages. However, there will be only a single stream to the data server. This optimization saves both CPU resources and network bandwidth.

1.4.7 Logging and Configuration Management

To aid developers with troubleshooting and debugging, the API supports logging of error and informational messages to the Microsoft Visual Studio debug console. Applications can request that an actual log file be opened and messages be directed to that file. In addition, applications using the API are allowed to log events to the same file

Applications typically need to "soft-code" certain application settings. For example, the hostname of the FactSet Data Server should be stored in some configuration file or system registry. The API includes functionality to assist applications in querying configuration files and system registries.

1.4.8 Threading Support

This API is thread-safe. This means that multiple threads are allowed to operate on the same object.

Chapter 2 Building Applications

2.1 Toolkit Organization

For Microsoft platforms, the toolkit is extracted from a simple Windows MSI file. The installation folder is specified during installation using a standard dialog (C:\Users*xxx*\AppData\Local\FactSet\DataFeed).

Directory/Filename	Contents	Additional Notes
RELNOTES.TXT	Contains the latest release notes for this version of the toolkit.	
VERSION.TXT	Contains the toolkit's version label and build number.	
CHECKSUMS.TXT	A file containing the check-sum of each file in the toolkit.	
LICENSES.TXT	Contains license agreements.	
bin/	Binary utilities and samples.	This directory also includes the actual debug and release DLL's.
etc/	Definition files	Example: rt_fields.xml
include/	The API header files	All the include/ files are in the FDS subdirectory
lib/	The API library files	Each platform will have its own folder (e.g., mswin). See the library naming convention section. A subdirectory exists for each supported platform.
log/	Sample programs and utilities will log to this directory.	Microsoft applications are able to use the registry to ensure logs get placed in this directory.
sample/	Sample applications	Each folder in this directory is targeted for a particular language (e.g., the APICOM_C# folder contains C# applications).

2.1.1 Library Naming Conventions

This toolkit includes four versions of the FactSet DataFeed COM library: a release version, named FDSRTCom.dll/FDSRTCom_x64.dll and a debug version, named FDSRTComd.dll/FDSRTComd_x64.dll.

2.2 Compiling Applications

The first step in compiling an application would be to install the toolkit using the provided Windows MSI. During the installation, the release DLL will be registered as a COM component called "FDSRTCom 2.2 Toolkit Type Library".

When using Microsoft development environments, the library must be added as a reference to use the toolkit. To simplify the following steps, assume that the root of this archive is located in *{FDS_ROOT}*. In the default case this would be set to "C:\Users*xxx*\AppData\Local\FactSet\DataFeed".[3]

[3] It is often better to use relative paths for toolkit installation directory (e.g. the sample applications in the toolkit use the relative path "..\..\..\" to locate header and library files).

2.2.1 Microsoft Visual Basic 6

Open the References dialog by selecting References… from the Project menu. Place a checkmark next to "FDSRTCom 2.2 Toolkit Type Library" in the Available References. If the library is not listed, add it to the list by using the Browse... button and selecting {FDS_ROOT}\bin\FDSRTCom.dll for the release DLL or {FDS_ROOT}\bin\FDSRTComd.dll for the debug DLL.

2.2.2 Microsoft Visual Studio

Open the Add Reference dialog by selecting Add Reference… from the Project menu. Select the COM tab. Locate the "FDSRTCom 1.0 Type Library" and double-click to add it to the Selected Components. If the library is not listed, add it to the list by using the Browse… button and selecting {FDS_ROOT}\bin\FDSRTCom.dll for the release DLL or {FDS_ROOT}\bin\FDSRTComd.dll for the debug DLL.

2.2.3 Other Applications

The table below lists the program identifiers needed to import the FDSRTCom Library into other applications.

Program Identifier (progid)	Class
FDSRTCom.FDF	FDF
FDSRTCom.RT_Message	RT_Message

2.3 Running Applications

Dynamic libraries need to be installed on any system that will execute applications built using the API. It is the responsibility of the application developer to ensure these libraries are available on all run-time systems.

The FactSet DataFeed COM library must be registered with the operating system prior to runtime. To do this, run regsvr32 "{FDS_ROOT}\bin\FDSRTCom(_x64).dll" (for the release DLL) or regsvr32 "{FDS_ROOT}\bin\FDSRTComd(_x64).dll" (for the debug DLL).

In addition, the COM library must be able to locate the fdsrt DLL. Microsoft has a well-defined search order for applications that need to locate a DLL. The exact reference article can be found at https://msdn.microsoft.com/en-us/library/windows/desktop/ms682586(v=vs.85).aspx.

In most cases, applications will search the directory in which the application is loaded, then the system directories (including the Windows systems directory), and finally all directories listed in the PATH environment variable.

Application developers that use the FactSet DataFeed COM API should ensure that the fdsrt DLL is available and locatable on all run-time systems. It is common to place this DLL in the same directory as the application.

In order to connect to the local FactSet workstation and use it as a data source, the current minimum supported or a more recent version of the FactSet workstation must be installed on the user's machine.

Chapter 3 Programming with the API

3.1 Program Setup and Initialization

3.1.1 Standard Conventions

This API is designed so that its interfaces adhere to a common set of standards. The following conventions are used by the FactSet COM API:

- All methods are lower case with the '_' character to separate words.
- Some methods have optional parameters with default values. These parameters are denoted by brackets (e.g. [*snapshot_only As Boolean* = False]) in the class overviews. Optional parameters do not apply to languages which do not support them, such as C#.

3.1.2 Exceptions

Methods that need to report an error do so by throwing an exception. The exceptions that are thrown contain a description of the error encountered in this format: "Interface: description of error". Below are some examples of retrieving this information.

[Visual Basic 6.0]

```
Private Sub Form_Load()
      Set fdf = New FDF

      On Error GoTo error_load_map
      fdf.load_field_map "nonexistant.xml"
      Exit Sub

error_load_map:
      MsgBox Err.Description
End Sub
```

[C#]

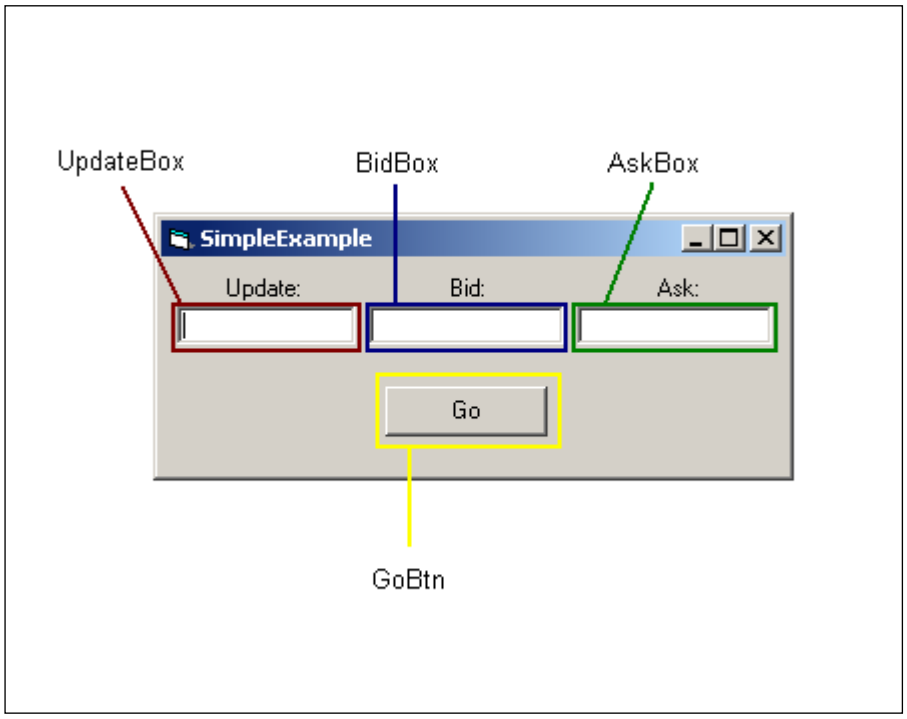
```
static void Main()
{
     fdf = new FDFClass();

      try
      {
          fdf.load_field_map("nonexistant.xml");
```

```
        }
        catch (Exception e)
        {
            MessageBox.Show(e.Message);
        }
}
```

In both examples, a message box will pop up declaring "IFDF: error loading field map from file nonexistant.xml".

3.1.3 A Complete Example

MainForm Layout:



Visual Basic Code:

```
Public WithEvents fdf As FDF

Private Sub fdf_OnMessage(ByVal tag As Long, ByVal msg As IRT_Message)
    If msg.is_error Then
        MsgBox "Error: " + msg.error_description
        fdf.Cancel tag
        Exit Sub
    End If

    Dim update As String
    update = msg.get(FIDS.MSG_TYPE)
    Dim bid As String
    bid = msg.get(FIDS.BID_1)
    Dim ask As String
    ask = msg.get(FIDS.ASK_1)

    UpdateBox.Text = update
    BidBox.Text = bid
    AskBox.Text = ask

    ' if the server closed the stream close our side as well
    If msg.is_closed Then
        fdf.Cancel tag
    End If

End Sub

Private Sub Form_Load()
    Set fdf = New FDF

    fdf.load_field_map "..\..\..\etc\rt_fields.xml"
    fdf.connection_info = "client:secret@api-stage.df.factset.com"
End Sub
```

```
Private Sub GoBtn_Click()
    On Error GoTo error_connect
    fdf.Connect

    Dim tag As Integer
    ' create a real-time request for service=FDS1, symbol=FDS-USA
    tag = fdf.request("FDS1", "FDS-USA")
    MsgBox "Made a request for FDS-USA. Tag = " + Str(tag)

    Exit Sub

error_connect:
    MsgBox "Unable to connect: " + Err.Description, vbOKOnly, "Error"
End Sub
```

3.2 Connecting to a Data Source

An application connects to a data source during initialization. There are two options when picking a data source to connect to: a FactSet Data Server and the local FactSet workstation. A connection to a FactSet Data Server occurs over the Internet or a WAN via TCP/IP. A connection to the local FactSet workstation occurs on the user's local machine. When connecting to a Data Server, applications should set the connection information, and then call the connect() function. . There are two ways to authenticate the user. One is basic authentication and the other is OTP authentication. OTP authentication is done by passing the credentials to set_connnection_info function. To connect to the local FactSet workstation, only a call to the workstation_connect() function is required[4].

By default, connect() and workstation_connect() are synchronous, and in rare cases a call may block for an extended period of time (currently set to 60 seconds). If applications wish to use a non-blocking connect, true should be passed as to the async parameter of the connect functions.

The host for production data is api.df.factset.com for production and api-stage.df.factset.com for beta. If canned data is required for development purposes the host canned-stage.df.factset.com with the FDS_C service should be used.

```
' connect to api.df.factset.com with user="client" and password="secret".
fdf.connection_info = "client:secret@api-stage.df.factset.com"

fdf.Connect              ' connects synchronously
fdf.Connect True         ' connects synchronously
fdf.Connect False        ' connects synchronously

' connects synchronously to the FactSet workstation
fdf.workstation_connect("CLIENT-1234")
```

[4] The workstation_connect() and the FactSet workstation need to be run under the same Windows user or FactSet will give an error that there are more than one instance of Marquee running.

```
' connects synchronously to the FactSet workstation
fdf.workstation_connect("CLIENT-1234", "", False)
' connects asynchronously to the FactSet workstation
fdf.workstation_connect("CLIENT-1234", "", True)
' connects to datafeed server with OTP authentication
fdf.set_connection_info("api-stage.df.factset.com",   "client",   "AAAA",   "5c706e...",
"730332...", "C:\\Path\\To\\Counterfile", true);
```

A synchronous connect operation will block until both the TCP connection is established and the application has successfully authenticated with the data source. If a synchronous connect operation fails, applications must do one of the following:

1. Retry the connect operation at some future time.
2. Connect asynchronously.
3. Exit the application.

An asynchronous connect operation will return immediately. If an asynchronous connect operation throws an exception, a connection will never get established. In this case, the application should log the error and exit. This is a rare condition which will only happen if an operating system resource could not be created (like a thread).

Upon returning from a successful asynchronous connect operation, the connection and authentication will be processed by an API thread. The OnControl event will be fired after a successful or unsuccessful connect. If the connection fails, the connection is retried periodically[5].

❖ ***If connect() or workstaion_connect() throw an exception, the connection will never get established. Applications must issue a successful connect before dispatching any messages. This behaviour is true for both asynchronous and synchronous connections. However, applications are allowed to make requests before a connection is established. These requests are queued internally within the API until a successful connection is established.***

3.2.1 Connection Strings

Currently both basic authentication and One Time password[6] is valid authentication methods but users are being migrated to OTP.

One time Password – set_connection_info

set_connection_info ("apt-stage.factset.com", "client", "AAAB", NULL, NULL, " C:\\Path\\To\\Counterfile", true); - The API will connect to the host "api-stage.factset.com" on the default port of "6681". It will use a username of "client" and a One Time password generated by the key and counter as per 3.2.1.1 using the Key ID AAAA and the key and counter file C:\\Path\\To\\Counterfile.

Basic Authentication connection_info

[5] If the connection is terminated (via a TERMINATE control message), connection attempts will no longer be retried. This is typical when the user credentials are invalid (see Appendix B).
[6] FactSet leverages the HMAC-Based One-Time Password Algorithm described in RFC 4226 (http://www.ietf.org/rfc/rfc4226.txt) and session tokens to ensure all requests to the API are made by authenticated users.

In order to connect to the FactSet Data Server, the application must set the host name (or IP address), the port number, the username, and the password. These items should be passed into the `connection_info` property. The property takes a string as input. The following outlines some examples.

connection_info = "client:aaa@api-stage.factset.com" – The API will connect to the host "api-stage.factset.com" on the default port of "6681". It will use a username of "client" and a password of "aaa".

connection_info = "client@10.2.4.5:4063" – The API will connect to the host 10.2.4.5 on port 4063 using the username "client". The password is empty in this case.

connection_info = "" – The API will use the global property RT_CONNECTION from the FDF class. The format of the RT_CONNECTION string is explained in Appendix C and is identical to that of the previous examples (e.g. client@10.2.4.5:4063).

connection_info = "reg:/HKEY_LOCAL_MACHINE/Software/FactSet/FDF" – The API will look for a property named RT_CONNECTION in the Windows registry. The hive location is HKEY_LOCAL_MACHINE, and /Software/FactSet/FDF is the path within the hive.

connection_info = "file:/etc/connection_info.cfg" – The API will look for a property named RT_CONNECTION in the file \etc\connection_info.cfg. The format of this file is given in Appendix D.

Setting the `connection_info` property will only throw an exception if the host could not be extracted given the specified uri. It does not check if the host is valid, or if the host can be translated to an actual IP address. It will simply store the connection information for later use. The connect() method will later use this information to resolve the hostname and port before attempting the connection to the FactSet Data Server.

3.2.1.1 Retrieving the One Time Password[7]

The authentication protocol for Exchange DataFeed is using One Time password. At the initial setup the key administrator[8] will need to follow the below steps to generate the key and counter required to authenticate with OTP.

1. Go to http://auth-setup.factset.com
2. Login using the FactSet .NET account received in the welcome email.
3. Enter the serial number tied to the server account used to connect to the feed.
4. Make sure the PROD is selected, rather than BETA.
5. Click Get New Key.
6. Create a new file - On the first line, copy and paste the "Key" from the web site (don't include the word "Key:", just the actual string).
7. On the second line, copy and paste the counter value.
8. Save this file as <KeyId>.data. Most likely that will be "AAAA.data" and use this file as input in the set_connection_info function as per below.
9. Alternatively take note of the values and use directly in set_connection_info.

[7] All users will migrate to OTP in the future
[8] The key administrator needs to be given access to be able to generate the key, contact your FactSet representative to get the required access enabled.

3.2.1.2 Connect with OTP

In the below samples three different examples of how to use the key and counter extracted above is used in `set_connection_info`.

```
// Connect to api-stage.df.factset.com with user="client" with key/counter file
// C:\Path\To\Counterfile\AAAA.data contains the key (hex string) on the first
// line and the counter (decimal format) on the second, for user "client" and // device
ID "AAAA"
fdf.set_connection_info("api-stage.df.factset.com", "client", "AAAA", NULL, NULL,
"C:\\Path\\To\\Counterfile", false);
```

```
// Connect to api-stage.df.factset.com with user="client" with key/counter file, // or
given values if no file exists
// If C:\Path\To\Counterfile\AAAA.data contains a key and counter, those will be // used
instead of the given key "5c706e..." and counter "730332..."
// Otherwise the given key/counter will be used and // C:\Path\To\Counterfile\AAAA.data
will be created from the given values to be
// used for subsequent attempts
fdf.set_connection_info("api-stage.df.factset.com", "client", "AAAA", "5c706e...",
"730332...", "C:\\Path\\To\\Counterfile", false);
```

```
// Connect to api-stage.df.factset.com with user="client" with given values
// regardless of existing key/counter file.
// The given key/counter will be used and C:\Path\To\Counterfile\AAAA.data will be //
overwritten or created from the given values to be used for subsequent attempts
fdf.set_connection_info("api-stage.df.factset.com", "client", "AAAA", "5c706e...",
"730332...", "C:\\Path\\To\\Counterfile", true);
```

3.2.2 Synchronous Connect Sequence Diagram

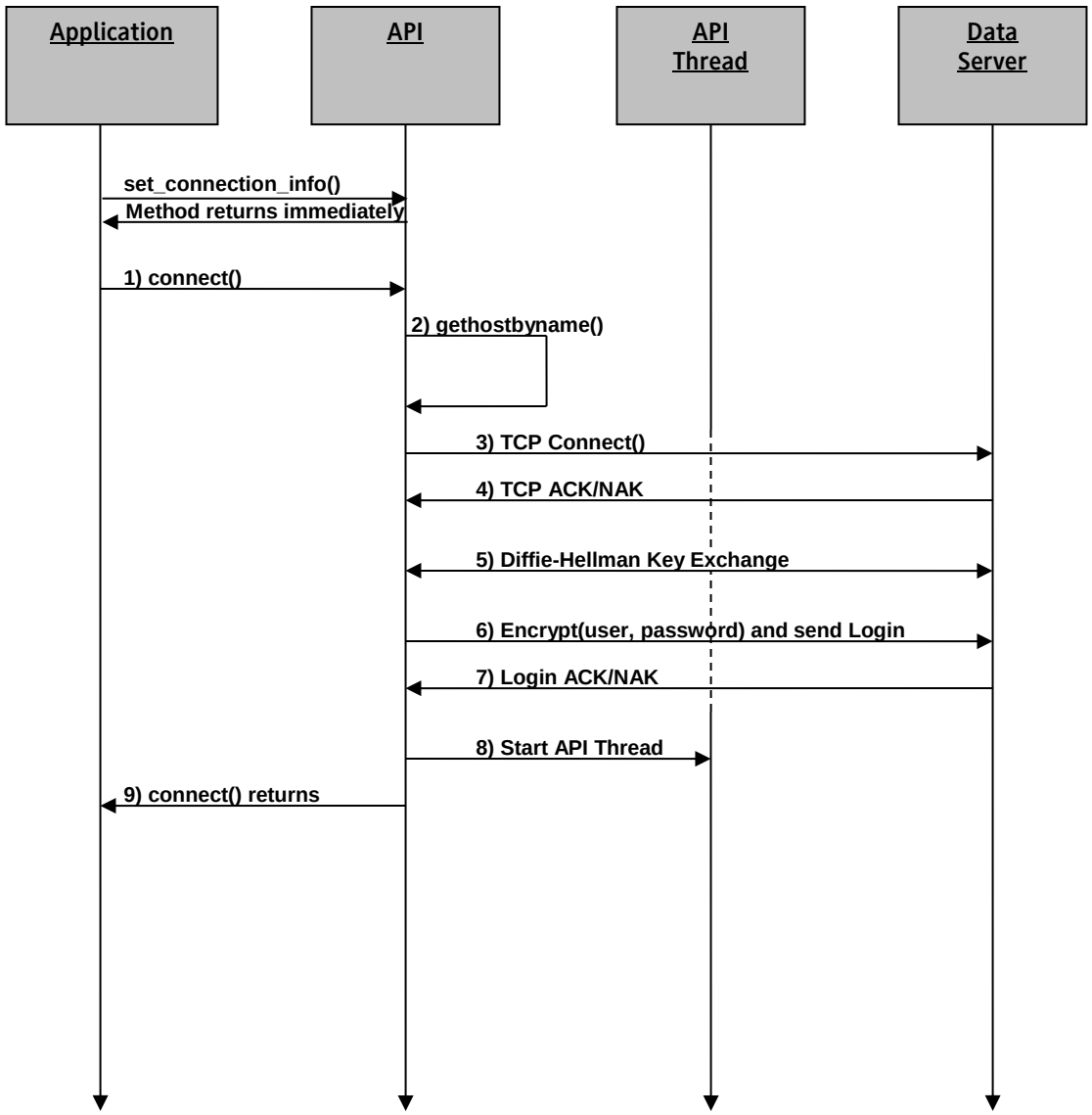


Figure 3: Synchronous Connect Sequence Diagram

3.2.3 Synchronous Connect Example

```
Public WithEvents fdf As FDF

Private Sub Form_Load()
    Set fdf = New FDF

    fdf.load_field_map "..\..\..\etc\rt_fields.xml"

    fdf.connection_info = "client:secret@api-stage.df.factset.com:6681"

    On Error GoTo error_connect
    fdf.connect

    ' …
    ' make requests
    ' …

error_connect:
    MsgBox "Unable to connect", vbOKOnly, "Error"
End Sub
```

The example code above demonstrates how to connect to the Data Server synchronously.

The first step in many programs would be to load a Field Map file. This file is located in the etc directory of the toolkit. The code above uses a relative path location ("..\..\..\etc\rt_fields.xml") based on the application's working directory. Applications should ensure that this path is correct (load_field_map() throws an exception if the file could not be located or opened). This method allows the application to translate field names to FIDs and vice versa. Although this step is not absolutely necessary, it helps with debugging and troubleshooting. For more information on the load_field_map method, see section *4.2.2 Field Translation.*

The second step is to set the connection information (i.e., host = api-stage.df.factset.com, port = 6681, user = client, password = secret). After setting the connection information, the application calls connect() to attach to the Data Server. For additional details on setting the connection information, see section *3.2.1 Connection Strings* or section *4.2.1 FDF Class.*

3.2.4 Asynchronous Connect Sequence Diagram

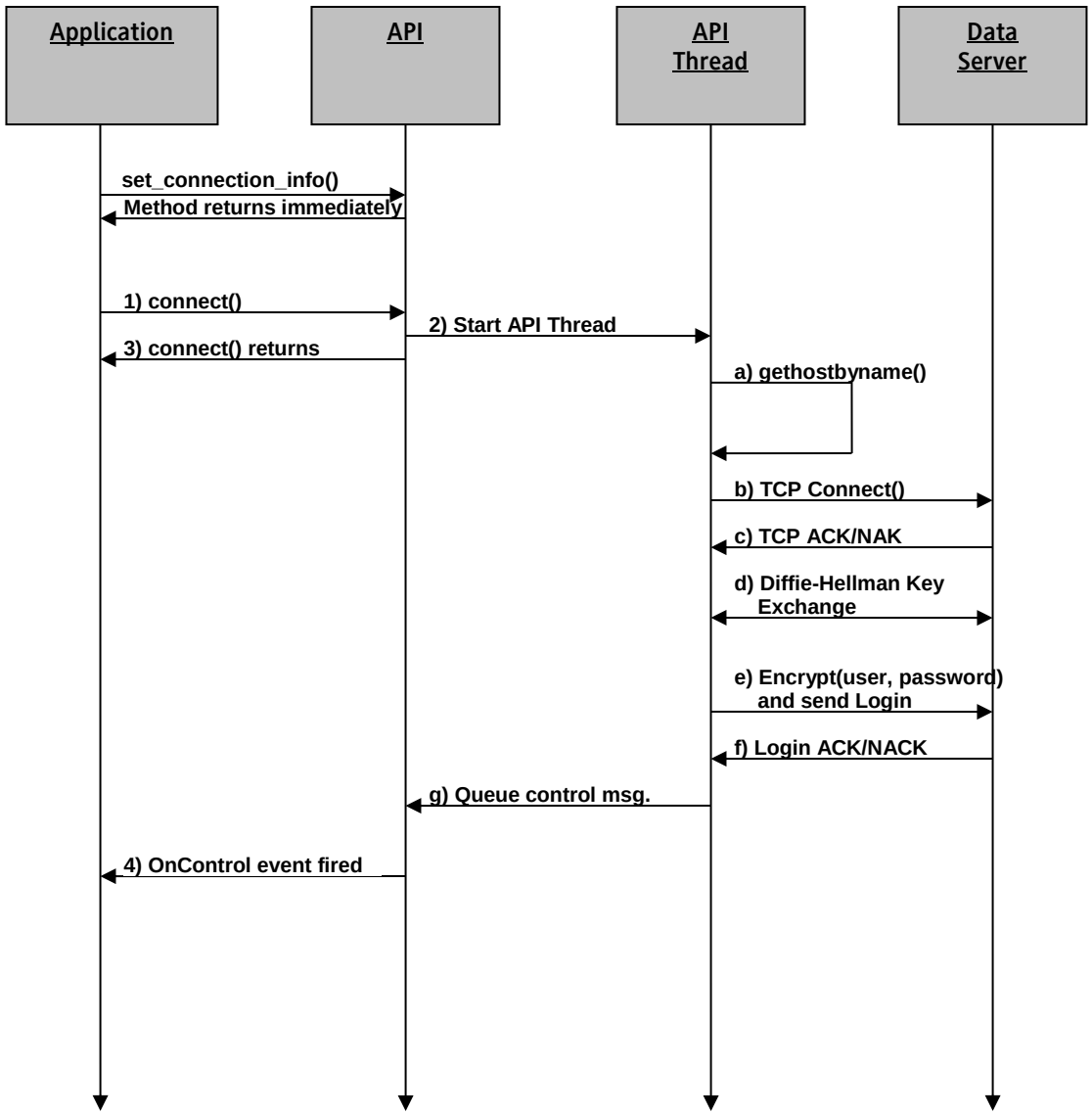


Figure 4: Asynchronous Connect Sequence Diagram

3.2.5 Asynchronous Connect Example

```
Public WithEvents fdf As FDF

Private Sub Form_Load()
    Set fdf = New FDF

    consumer.load_field_map ("..\..\..\etc\rt_fields.xml")

    fdf.connection_info = "client:secret@api-stage.df.factset.com:6681"

    On Error GoTo error_connect
    fdf.connect True     ' connect asynchronously

    ' …
    ' make requests
    ' …

error_connect:
    MsgBox "Unable to connect", vbOKOnly, "Error"
End Sub
```

The example code above demonstrates how to connect to the FactSet Data Server asynchronously. The code is identical to the previous example except for an additional parameter passed into connect().

3.3 Requests and Cancels

3.3.1 Opening the Stream

Requests are made using the `FDF.request()` method. Although requests are typically made after connection establishment, the application can make requests at any time. If the API is disconnected from the server, or a particular service is not available, requests will be queued internally by the API. The request method is defined as follows:

[Visual Basic 6.0]
```
Public Function request(service As String, key As String,
                        Optional ByRef is_pending As Boolean,
                        Optional snapshot_only As Boolean = False,
                        Optional symbol_type As SymbolType = SYMBOL_TYPE_UNKNOWN,
                        Optional options As String) As Long
```

[C#]
```
public int request(string service, string key, bool snapshot_only,
                   out bool is_pending, SymbolType symbol_type, string options)
```

The service and the key are the first two parameters required by the request() method. A service is a string that identifies a data source and the symbol is the key for that particular data source. In addition, the method allows applications to explicitly set the snapshot flag to true for a static request and false for a dynamic request. A dynamic request will open a virtual stream with the Data Server for that particular data element. A static request will also open a virtual stream, but the first message on that stream will indicate a closure of that stream. This type of request is typically called a snapshot request.

On each call to FDF.request() a tag will be returned. ***The method will NOT throw an exception if the service is unavailable or there is no connection. Instead, a valid tag will be returned (since the request has been queued) and the out parameter is_pending will be set to true.*** This tag is an integer and is the resource id for the stream that has just been opened. A tag is returned for both static and dynamic requests.

3.3.2 Closing the Stream

The messages for a stream will be passed to an event handler along with the stream id. Eventually the stream should be closed and the resource tag freed. This resource can be freed in either two ways: 1) calling `FDF.disconnect()` or 2) calling `FDF.cancel(tag)`. The stream will continue to be open until one of these two functions is called. This is true even for snapshot requests. As mentioned before, snapshot data is treated as a request for a single message, and that message should have the closed (end of stream) property set to true. This indicator tells the application that the stream is closed on the server-side. It is the responsibility of the application to make sure the tag is cancelled after receiving the snapshot message. A call to cancel() will close the stream on the client-side.

> ❖ ***Canceling a tag that was already cancelled results in undefined behavior. Leaking tags can cause applications to consume more memory and respond slower. Applications should treat tags like they treat open files, or pointers to heap-allocated memory.***

3.3.3 Tag Ownership and Lifetime

Tags are assigned by the API and given to clients. The lifetime of every tag is controlled by the application. Applications create tags via the request() method and free the tag via the cancel() method. Clients cannot choose the tag identifier. They are assigned by the API.

3.3.4 Dynamic Request

```
Public WithEvents fdf As FDF

Private Sub fdf_OnMessage(ByVal tag As Long, ByVal msg As IRT_Message)
    If msg.is_error Then
        MsgBox "Error: " + msg.error_description
    End If

    MsgBox "Message: " + msg.to_string

    ' if the server closed the stream close our side as well
    If msg.is_closed Then
        fdf.Cancel (tag)
    End If

End Sub

Private Sub Form_Load()
    Set fdf = New FDF

    ' set up connection (see previous code)

    Dim tag As Integer
    ' create a real-time request for service=FDS1, symbol=FDS-USA
    tag = fdf.request("FDS1", "FDS-USA")

    MsgBox "Made a request for FDS-USA. Tag = " + Str(tag)
End Sub
```

The example code above shows a request being made for the symbol "FDS-USA" under the "FDS1" service. Since the snapshot flag parameter is false (by default), the request is for a dynamic subscription. The resource id for the stream is returned to the local variable tag.

The event handler simply pops up a message box containing the message. However, it does check to see if the stream was closed, and if so, it closes the client-side stream by canceling the tag. The server may close the stream at any time. In addition, error messages (e.g., RT_E_NOT_FOUND) will cause the stream to set the close/end-of-stream indicator. The example event handler handles both of these conditions.

3.3.5 Static Request

```
Public WithEvents fdf As FDF


Private Sub fdf_OnMessage(ByVal tag As Long, ByVal msg As IRT_Message)
    MsgBox "Update: " + msg.get(FIDS.MSG_TYPE)


    ' No reason to check is->closed(), since we only want a single
    ' message.  So just call cancel() after processing
    fdf.Cancel (tag)
End Sub


Private Sub Form_Load()
    Set fdf = New FDF


    ' set up connection (see previous code)


    Dim tag As Integer
    ' create a static request for service=FDS1, symbol=FDS-USA
    ' The parameter "True" sets the snapshot_only flag
    tag = fdf.request("FDS1", "FDS-USA", True)


    MsgBox "Made a request for FDS-USA. Tag = " + Str(tag)
End Sub
```

The request for snapshot data is similar to the one for dynamic data except that the snapshot parameter is set to true. In fact, the event handler from the previous example could have been used in this example. Since static requests will close the stream on the first message, the previous event handler would have cancelled the tag. However, the current example states the application's intentions more clearly when the call to FDF.cancel() is explicit (like above), rather than based on a predicate (like the previous example).

3.3.6 Canceling Requests

Applications cancel the request using the tag given at the time of request. Applications can cancel the tag at any time (even before receiving the first message).

Once a tag is cancelled the integer identifier can be reused. This is typical, and it is common for the next request to be given the exact same tag as a previously cancelled stream. However, this "new tag" identifies a "new stream." The API guarantees that messages from the previous stream cannot be delivered to this new stream.

3.4 Processing Events

In order to fire events to the application defined event handlers, a Windows event loop must be present in the application. All GUI threads contain the required event loop by default. If a console application is desired, the main thread must contain an event loop. This is best accomplished using Application.Run(), as demonstrated in the C# example code in Appendix A.

3.5 Processing the Messages

3.5.1 FID Value Pairs

The API makes heavy use of the widely accepted standard of representing data as field/value pairs. This self-describing data structure tags all data elements with an integer identifier (FID or field identifier).

The value is a string. Every field/value pair has an agreed-upon meaning by both the data sources and the consuming applications. This meaning can never be changed once published to the applications.

3.5.2 Field Identifiers

The current field identifiers are available through two sources. The first is the `FIDS` enumeration. This enumeration defines human-readable static constant integers for the current list of known field ids. This is the usual method of identifying a field by name in actual code.

The second file is rt_fields.xml (also included in the toolkit). This file can be loaded by the FDF.load_field_map() method and allows applications to translate human readable names to field ids at runtime (as opposed to compile time using the `FIDS` enumeration).

3.5.3 Messages

All requests open a stream, which is a virtual tunnel of messages. A message has certain properties such as a service, a key, and some other flags (see section *4.3.1 RT_Message Class* for more information).

An RT_Message is simply a container of fields (i.e., fids and values). The fields can be extracted using the methods defined in the RT_Message class (see section *4.3.1 RT_Message Class* for more information).

3.5.4 Processing a Message Example

```
Public WithEvents fdf As FDF

Private Sub fdf_OnMessage(ByVal tag As Long, ByVal msg As IRT_Message)
    If msg.is_error Then
        MsgBox "Error: " + msg.error_description
        fdf.Cancel tag
        Exit Sub
    End If

    Dim update As String
    update = msg.get(FIDS.MSG_TYPE)
    Dim bid As String
    bid = msg.get(FIDS.BID_1)
    Dim ask As String
    ask = msg.get(FIDS.ASK_1)

    UpdateBox.Text = update
    BidBox.Text = bid
    AskBox.Text = ask

    ' if the server closed the stream close our side as well
    If msg.is_closed Then
        fdf.Cancel (tag)
    End If

End Sub

Private Sub Form_Load()
    Set fdf = New FDF

    ' set up connection (see previous code)

    Dim tag As Integer
    tag = fdf.request("FDS1", "FDS-USA")
```

```
    MsgBox "Made a request for FDS-USA. Tag = " + Str(tag)
End Sub
```

The example code above shows one way to process a message from an event handler. The event handler simply fills in the message type along with the bid and ask into the appropriate text boxes. In addition, it checks to see if the stream was closed, and if so, it closes the client-side stream by canceling the tag.

The server may close the stream at any time. In addition, error messages (like RT_E_NOT_FOUND) will cause the stream to set the close/end-of-stream indicator. The example event handler handles both of these conditions.

Chapter 4 API Class Reference

4.1 API Constants

4.1.1 Message Errors

When an RT_Message has the `error` property set to true, the error is conveyed to the application via the `rt_errno` enumeration. The list of the possible errors is noted below.

```
typedef enum rt_errno {
  RT_NO_ERROR    =  0,   // All is good...
  RT_E_START     = -50,  // Start of RT errors
  RT_E_UNKNOWN   = -51,  // Unknown error
  RT_E_NO_SERV   = -52,  // No Service Available
  RT_E_NOT_FOUND = -53,  // Field, or Record not found
  RT_E_RENAME    = -54,  // Record has been renamed
  RT_E_TIMEDOUT  = -55,  // Operation Timed Out
  RT_E_EXISTS    = -56,  // Already exists
  RT_E_LIMIT     = -57,  // Maximum application limit has been reached
  RT_E_PROTOCOL  = -58,  // Any Protocol error (message, file format, network)
  RT_E_INVAL     = -59,  // Invalid parameter to method call
  RT_E_RESOURCE  = -60,  // Operating system resource exhausted
  RT_E_NO_CONN   = -61,  // No connection to the server
  RT_E_VERSION   = -62,  // Incorrect version
  RT_E_SHUTDOWN  = -63,  // User has shutdown the system
  RT_E_ACCESS    = -64,  // Permission denied
  RT_E_END       = -65   // End of RT errors
} rt_errno;
```

4.1.2 Field Identifiers

Field identifiers are integers that can be used to index into either the `RT_Message` object. The list of field identifiers is available through the `FIDS` enumeration, which can be viewed using the Microsoft Visual Studio Object Browser. Applications that need to reference field identifiers by a symbolic name can do so by using this enumeration.

4.2 FDF

The FDF class manages the connection to the FactSet Data Server. It is the class used to connect, request data, cancel subscriptions, manage the event loop, and finally disconnect. In addition, the FDF object contains several methods that manage field names, configuration properties and logging. This object is the heart of the FactSet DataFeed COM API. For the sake of simplicity, the object's properties, methods and events are outlined in the style of Visual Basic 6.0.

Properties

Property **connection_info** As String

Methods

Sub **connect**([*async_connect As Boolean* = False])

Sub **workstation_connect**([*user As String, [passwd As String], [async_connect As Boolean* = False])

Sub **disconnect**([*keep_registration As Boolean* = False])

Function **is_connected**() As Boolean

Function **request**(*service As String, key As String,* [*snapshot_only As Boolean* = False], [*options As String*]
[*ByRef is_pending As Boolean* = False]) As Integer

Sub **cancel**(*tag As Integer*)

Sub **load_field_map**(*filename As String*)

Function **get_field_name**(*fid As Integer*) As String

Sub **load_properties**(*uri As String, [append As Boolean* = False])

Function **get_property**(*name String,* [*def_val As String*]) As String

Sub **set_property**(*name As String, value As String*)

Sub **log_open**(*file As String,* [*append As Boolean* = False])

Sub **log_close**()

Function **log_level**(*new_level As RT_LogLevel*) As RT_LogLevel

Sub **log**(*severity As RT_LogLevel, msg As String*)

Function **get_services**() As RT_Message

Events

Event **OnMessage**(*tag As Integer, msg As RT_Message*)

Event **OnControl**(*is_connected As Boolean, msg As RT_Message*)

4.2.1 The FDF Class

The `FDF` class manages the connection to the FactSet Data Server. Applications will use this object to open a connection to the FactSet Data Server and request information. This object will manage all the subscriptions on behalf of the application.

Event Handling

The application can receive two types of events from the FDF object. The first type of event is **OnMessage**:

[Visual Basic 6.0]

```
Private Sub fdf_OnMessage(ByVal tag As Long, ByVal msg As IRT_Message)
```

[C#]

```
private void fdf_OnMessage(int tag, RT_Message msg)
```

OnMessage is used for application messages to items requested. Applications should define an event handler according to the above method signature to receive messages for requests. The parameters are as follows:

- ***tag*** is the tag that was returned when the orginal request was made. It is the resource that identifies this stream, and is needed to cancel the stream.
- ***msg*** is the actual update (or initial) message from the server. A dynamic open stream will deliver a series of messages. The initial message will contain all the available fields, and the updates will contain only those fields that need to be changed.

The second type of event, **OnControl**, is a control notification:

[Visual Basic 6.0]

```
Private Sub fdf_OnControl(ByVal is_connected As Boolean,
                          ByVal msg As IRT_Message)
```

[C#]

```
private void fdf_OnControl(bool is_connected, RT_Message msg)
```

Applications that want to receive control messages should define an event handler with the above method signature to receive these messages. Control messages will typically indicate events such as Connected, Disconnected, Service Attachment, and so on. For a complete list of control messages, see the Appendix located at the end of this document. The first parameter (`is_connected`), indicates if the current connection is valid. The second RT_Message parameter has information such as the type of control notification.

Control messages are used to dynamically inform applications of various events such as the following:

- Removal of a service – When a service is removed and no longer available for requests, a control message will be sent to the application.

- Addition of a new service – When a new service has attached and is ready to accept new connections a control message will be sent to the application.
- TCP connection termination – When a connection socket is terminated (for any reason), a control message will be sent. The reason for termination will be included in the control message.
- TCP connection notification (for asynchronous connections) – When the application is using asynchronous connections, the API will deliver a CONNECTED control message upon a successful TCP connect.
- For complete details on the type of control messages that can be received, as well how each one should be handled see Appendix B.

Setting the connection information

The **connection_info** property is used to set the connection information. In order to connect to the FactSet Data Server, the application must set the host name (or IP address), the port number, the username, and the password. The property takes a string as input. The following outlines some examples.

connection_info = "client:aaa@api-stage.factset.com" – The API will connect to the host "api-stage.factset.com" on the default port of "6681". It will use a username of "client" and a password of "aaa".

connection_info = "client@10.2.4.5:4063" – The API will connect to the host 10.2.4.5 on port 4063 using the username "client". The password is empty in this case.

connection_info = "" – The API will use the global property RT_CONNECTION from the FDF class. The format of the RT_CONNECTION string is explained in Appendix C and is identical to that of the previous examples (e.g. client@10.2.4.5:4063).

connection_info = "reg:/HKEY_LOCAL_MACHINE/Software/FactSet/FDF" – The API will look for a property named RT_CONNECTION in the Windows registry. The hive location is HKEY_LOCAL_MACHINE, and /Software/FactSet/FDF is the path within the hive.

connection_info = "file:/etc/connection_info.cfg" – The API will look for a property named RT_CONNECTION in the file \etc\connection_info.cfg. The format of this file is given in Appendix D.

The **connection_info** property will only throw an exception if the host could not be extracted from the specified uri. It does not check if the host is valid, or if the host can be translated to an actual IP address. It will simply store the connection information for later use. The **connect()** method will later use this information to resolve the hostname and port before attempting the connection to the FactSet Data Server.

Getting the connection information

The **connection_info** property can also be used to retrieve the connection information. It will contain a string in the form of **USER:PASSWD@HOST:PORT**.

❖ ***Retrieving from the connection_info property does NOT return the same string passed into setting the connection_info property. Setting the connection_info property can take a URI resource. This resource can identify a file, registry location, or the connection string itself. Instead, retrieving the connection_info property returns the resolved user, password, host, and port information from the setting the connection_info URI.***

Connecting to a FactSet server

[Visual Basic 6.0]
```
Public Sub connect(Optional async_connect As Boolean = False)
```

[C#]
```
public void connect(bool async_connect)
```

The connect() function connects the API to a FactSet server over an Internet or WAN connection. The connection info must be set through the conneciton_info property.

Connecting to a local FactSet workstation

[Visual Basic 6.0]
```
Public Sub workstation_connect(user As String, Optional passwd As String,
                               Optional async_connect As Boolean = False)
```

[C#]
```
public void workstation_connect(string user, string passwd, bool async_connect)
```

The workstation_connect() function connects the API to the FactSet workstation program running on the current machine. In the default configuration, the user parameter should be the username and serial number of the client separated by a dash (e.g. "USER-1234") and the password should be a blank string. It is not necessary to set the connection info when using the workstation_connect() function.

Asynchronous vs. synchronous connect calls

When `async_connect` is `false`, the connect and workstation_connect methods are synchronous operations, and thus, will block until a valid connection is established. If the method does not throw an exception, applications can assume that the connection is valid. If an exception is thrown, the connection attempt has failed. Applications should retry the connect operation at sometime in the future or exit. Applications can issue requests before a successful connect().

Applications that wish to connect asynchronously (i.e., non-blocking), should explicitly pass in `true`. When the `async_connect` parameter is set to `true`, the connect methods will return immediately. If an exception is thrown, the asynchronous connect has failed[9]. If an asynchronous connect operation does not throw an exception, the connection is in progress. The OnControl event will be fired upon a successful or unsuccessful connect.

Disconnecting and querying the connection status and available services

[Visual Basic 6.0]
```
Public Sub disconnect(Optional keep_registrations As Boolean = False)
```

[C#]
```
public void disconnect(bool keep_registrations)
```

Disconnect will tear down the TCP connection to the server. Applications can specify whether the internal subscriptions should also be cleaned up. All subscriptions will be cancelled if `keep_registrations` is `false`. If the application wishes to keep the subscriptions, this call will generate "stale" message callbacks to all open streams. In this case, the events are fired during the call to disconnect.

[Visual Basic 6.0]
```
Public Function is_connected() As Boolean
```

[C#]
```
public bool is_connected()
```

[9] Applications should exit if an async connect(true) operation fails. This is an unrecoverable, serious error.

is_connected() simply returns the status of the connection. It is possible that a network or server condition can cause a TCP disconnect during normal operation. In this case, the API will attempt to retry the connection every so often. Immediately after a disconnect all open streams will receive stale messages. When the connection is re-established the open streams will be notified (via the event) with the refreshed non-stale data.

[Visual Basic 6.0]
```
Public Function get_services() As RT_Message
```

[C#]
```
public RT_Message get_services()
```

This method retrieves all the available services (a.k.a. data sources). The fid FIDS.SERVICE_NAME is appended to the returned RT_Message multiple times (one for each service available).

Requesting and canceling data streams

[Visual Basic 6.0]
```
Public Function request(service As String, key As String,
                        Optional snapshot_only As Boolean = False,
                        Optional options As String,
                        Optional ByRef is_pending As Boolean = False) As Long
```

[C#]
```
public int request(string service, string key, bool snapshot_only,
                   string options, out bool is_pending)
```

[Visual Basic 6.0]
```
Public Sub cancel(tag As Long)
```

[C#]
```
public void cancel(int tag)
```

request() and cancel() are the main entry points to open and close data streams. Applications call the request method to initiate a stream. The returned value is a tag. This is the resource that identifies the open stream. Leaking this tag will result in poor performance, and thus applications should manage this resource very carefully. Applications should cancel the tag using the cancel() method.

The request() function will ALWAYS return a resource tag and does not throw exceptions. For example, if the application requests a service which is not known, the is_pending out parameter is set to true. However, the request is still put in a queue, and when the service is attached, the request will be sent. Also, it is possible to issue requests on a disconnected system. In this case, the is_pending out parameter is set to true, and the request will be sent as soon as the connection is established.

❖ ***Every call to request() will generate a tag. The application is responsible for managing that tag. There are only two ways to free the resource: using the cancel() method or calling disconnect() with the default parameter.***

4.2.2 Field Translation

Field ids are simple integers. However, these integers are typically managed by an associated human-readable name. The `load_field_map` and `get_field_name` methods assist in translating names to ids and vice versa. Translating FIDs to names assists with debugging messages and also allows for human-readable configuration files. In essence, these methods allow names to be translated to numbers at run-time. If names are only required at compile time, these methods are not needed, and instead the application can just use the FIDS enumeration.

[Visual Basic 6.0]
```
Public Sub load_field_map(filename As String)
```

[C#]
```
public void load_field_map(string filename)
```

This method loads a field map from an XML file specified by filename. It will throw an exception if the file could not be opened.

4.2.3 Logging Within and Outside the API

The API logs errors to the Microsoft Visual Studio debug console. However, applications can open an actual log file if they desire. The following methods available from the FDF class assist with application logging.

[Visual Basic 6.0]
```
Public Sub log_open(file As String, Optional append As Boolean = False)
```

[C#]
```
public void log_open(string file, bool append)
```

log_open() opens a log file named by the parameter `file`. If the file exists and the append parameter is set to false, the old file is moved to a filename with a ".old" extension. All API log messages will now be logged to this file.

[Visual Basic 6.0]
```
Public Sub log_close()
```

[C#]
```
public void log_close()
```

log_close() closes the log file that was previously opened by log_open(). All log messages will now be directed to the debug console.

[Visual Basic 6.0]
```
Public Function log_level(new_level As RT_LogLevel) As RT_LogLevel
```

[C#]
```
public RT_LogLevel log_level(RT_LogLevel new_level)
```

By default all levels are logged. Applications can change the minimum level that will be logged. For example, a value of FDS::RT_LOG_ALL means that all levels will be logged, and a value of FDS::RT_LOG_NONE means nothing will be logged. A value of

RT_LOG_WARN will log all levels greater than or equal to RT_LOG_WARN (i.e. WARN, ERROR, and PANIC). The following logging levels are supported:

```
typedef enum RT_LogLevel {
            RT_LOG_ALL   =  0,
            RT_LOG_DEBUG =  1,
            RT_LOG_INFO  =  2,
            RT_LOG_WARN  =  3,
            RT_LOG_ERROR =  4,
            RT_LOG_PANIC =  5,
            RT_LOG_NONE  =  6
} RT_LogLevel;
```

[Visual Basic 6.0]
```
Public Sub log(severity As RT_LogLevel, msg As String)
```

[C#]
```
public void log(RT_LogLevel severity, string msg)
```

This method will allow any message to be logged to the log file[10].

4.2.4 Configuration Management

All applications need to "soft-code" certain application settings. For example the hostname for the FactSet Data Server should be stored in some configuration file or system registry. The API includes functionality to help manage application configuration files.

[Visual Basic 6.0]
```
Public Function get_property(name As String,
                             Optional def_val As String) As String
```

[C#]
```
public string get_property(string name, string def_val)
```

This method call will look up a property name and return the associated value. If the property is not found, the function will return **def_val**.

[Visual Basic 6.0]
```
Public Sub set_property(name As String, value As String)
```

[C#]
```
public void set_property(string name, string msg)
```

This method will set a property value given by name to a contents of value.

[Visual Basic 6.0]
```
Public Sub load_properties(uri As String, Optional append As Boolean = False)
```

[C#]
```
public void load_properties(string uri, bool append)
```

[10] When running debug builds using Microsoft Visual Studio, log messages will also be sent to the Output window.

This method loads the properties that are available in the given **uri**. If append is false, any pre-existing properties are removed. Otherwise, pre-existing parameters will remain, but may be overwritten. Appendix D explains the URI syntax as well as provides examples for reading a plain text file or the Windows registry.

Example Code:

```
  On Error GoTo error
  consumer.load_properties "etc\my_properties\cfg"

  Dim val As String
  val = consumer.get_property("MY_PROPERTY_NAME")
  MsgBox "MY_PROPERTY: " + val

error:
  MsgBox "Unable to open properties file"
```

The above example code loads a property file, “etc\my_properties.cfg”, and retrieves the property, “MY_PROPERTY_NAME”.

4.3 Messages

The `RT_Message` class represents all the messages in the system. The API delivers `RT_Message` objects to client event handlers. This class contains information applicable to all messages. For example, all messages will have a key, message flags, message state, and may also have associated permission information and/or an error condition. All RT_Messages contain a set of FID field pairs. A FID field is data that is identified by an integer.

Properties (Read-only)

Property **key** As String

Property **permissions** As String

Property **stale_reason** As String

Property **error_description** As String

Property **errno** As rt_errno

Property **initial** As Boolean

Property **update** As Boolean

Property **stale** As Boolean

Property **error** As Boolean

Property **closed** As Boolean

Property **complete** As Boolean

Property **count** As Integer

Property **empty** As Boolean

Methods

Function **get**(*fid As Integer*) As String

Function **get_by_idx**(*idx As Integer*, *fid As Integer*) As String

Function **exists**(*fid As Long*) As Boolean

Function **to_string**() As String

4.3.1 RT_Message Class

Object Creation

The RT_Message class cannot be instantiated directly. There are three ways to obtain an RT_Message object: through an OnMessage event, through an OnControl event and through the return value of the FDF.get_services() method.

RT_Message Interface

The following properties allow the application to query various pieces of information.

- key - is a string property which contains the key of the message.
- permissions - is a string property which contains the permission ID of the message.
- stale_reason - is a string property. Should be used if the message is stale (i.e. stale is true).
- errno - is an rt_errno property which contains the error code of an error message. For a complete list of possible errors see Appendix B.
- error_description - is a string property which describes any errors.
- stale - is a boolean property which is true for stale messages.
- error - is a boolean property which is true for error messages.
- closed - is a boolean property which is true when the stream is closed.
- initial - is a boolean property which is true for the initial message.
- update - is a boolean property which is true if the message is an update to an initial message (opposite of initial).
- empty - is a boolean property which is true if there are no FIDs in the message.
- count - is an integer property which contains the number of fids in the message.

The following methods are used to query field information.

[Visual Basic 6.0]
```
Public Function get(fid As Long) As String
```

[C#]
```
public string get(int fid)
```

This method returns a string containing the field data identified by the FID. If the FID is not found, an empty string is returned.

[Visual Basic 6.0]
```
Public Function get_by_idx(idx As Long, ByRef fid As Long) As String
```

[C#]
```
public string get_by_idx(int idx, out int fid)
```

This method returns a string containing the field data at a particular index. The index is zero-based to a maximum value of the count – 1. After the method returns, the variable passed in as fid will contain the FID of the field.

[Visual Basic 6.0]
```
Public Function exists(fid As Long) As Boolean
```

[C#]
```
public bool exists(int fid)
```

This method returns true if the FID exists in the message.

[Visual Basic 6.0]
```
Public Function to_string() As String
```

[C#]
```
public string to_string()
```

This method serializes the message into a human-readable string.

Chapter 5 Permission Service

FactSet has a permission system used to entitle its terminal users for real time or delayed exchange data for display-only-use, this system has been extended to enforce permissioning via third party integrators. By providing user permission maps, login status and an IP address check, the third party system can enforce FactSet's terminal permissions in their own system. This is called the Workstation Entitled API permission setup.

> ❖ ***Clients who subscribe to the Enterprise DataFeed and manage their own permissions and exchange re-distribution agreements do not need to use the Permission Service.***

5.1 Requirements

To use the Workstation Entitled API permission scheme, every user needs to have a unique FactSet Serial Number, either linked to a FactSet Workstation or to a FactSet Launch account. FactSet maintains the individual user's exchange permissions on their serial number. Exchange access through the third party terminal will be granted based on Serial Number access.

Every subscription to streaming data provided by FactSet contains a permission code. The third party system must match the permission code with the permission code contained in the user map of the user requesting the data. If there is a match, data can be passed on to that user. If there is no match, then the user is not entitled for the data and an error message should be displayed

In order for FactSet to comply with its exchange commitments, the third party must follow the instructions of the FactSet permission system. FactSet will audit any third party implementation to ensure its permissions are being enforced correctly.

The Permission service encapsulates all of the FactSet permission logic in to a simple ALLOW/DENY notification. Third parties must subscribe, listen, and follow all the permission statuses relayed by the permission service. The permission service generates data for each individual Factset user. So, the third party system must request and continue to listen to the permission service using the FactSet USERNAME-SERIAL combination. For example, XYZCOMPANY-12345 is passed to the permission service. Any changes will be sent via the service. In addition, the third party must provide an IP address or list of IP addresses. These two sets of information will be all FactSet needs to make a judgment on whether a user has access to data or is denied.

As a response, the permission service will provide two sets of information. One is the login status, represented by a 1 or 0. When the status is 1, the third party is allowed to send FactSet exchange data to its third party terminals. When the status is 0, no FactSet exchange data may be sent. The second is the permission map which is only available if the user is logged on. The third party must match the streaming data to the user's permission set to accurately permission the individual user.

> ❖ ***As mentioned above, the permission service is designed to provide streaming updates on the status of individual users. It is not necessary nor desirable to rapidly make new requests to this service in an attempt to discover changes because they will be streamed to the subscriber automatically.***

Only one subscription is allowed for a particular user, if the user attempts so authenticate on a second machine a "Duplicate subscription Error" will be sent to the first request. This is by design to signal that a user is already logged on from a different terminal. The correct behavior will be to allow the new login request and invalidate the original connection.

> ❖ ***FactSet provide utilities for firms that may want to check on the status of an individual user using the permission service. Because of the duplication subscription behavior, this wil shut down the induvidual in favor of the utility, which may not be the desired result.***

5.1.1 Authenticating with a FactSet Workstation

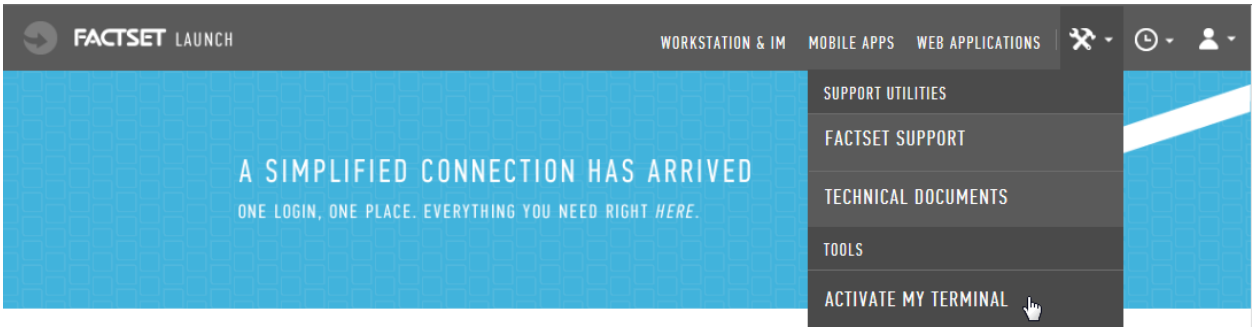
The user can only receive data while being logged into the FactSet workstation on the same machine as where the third party terminal software is running, this will be confirmed by an IP address check and logon status check.

If the third party terminal tried to run with the user not being logged in to FactSet, or logged in on a different machine, the third party terminal would fail the login test and would not receive any data.

5.1.2 Authenticating with FactSet Launch

FactSet Launch is a web portal where multiple FactSet services can be accessed through a single sign-on, the user's unique and permanent factset.net ID is used to login. The factset.net ID is linked to a FactSet Username and Serial Number with individual access to datasets and applications.

The Launch utility Activate my Terminal is available in the tools menu in FactSet Launch. The utility is collecting the local IP address from the machine where it is run to be used by the permission service. Activate my terminal is recommended to use through Chrome.

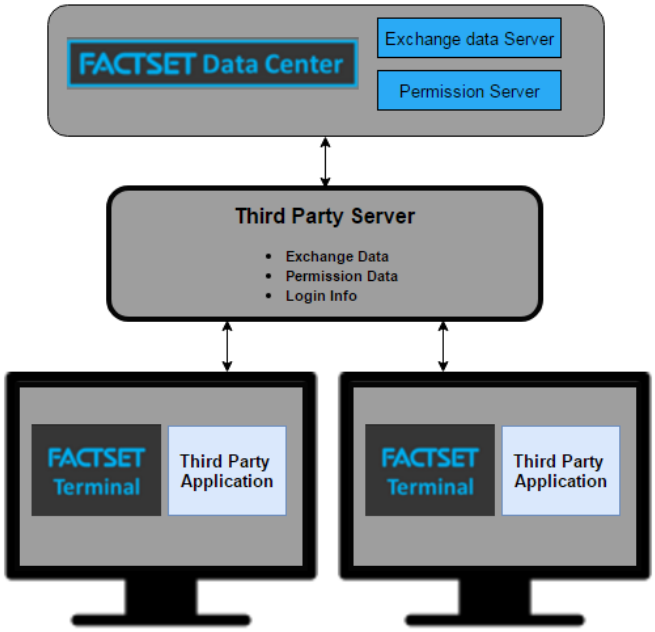


The user needs to authenticate through FactSet Launch on the same machine as the third party terminal is being used. This will be confirmed by an IP address check. Once authenticated access with be granted for 12 hours. After 12 hours the user needs to renew its access from launch.factset.com.

If the third party terminal is run without the user being authenticated through Launch in the last 12 hours, or authenticated on a different machine, the third party terminal fails the login test and will not receive any streaming data.

5.2 Workflow

An overview of the technology and workflow for this service is described below.



- **Step 1:** FactSet has a centralized system that manages all its end users' permissions, login status and Launch/Workstation IP address. A user logs into the FactSet terminal/Launch and the Permission Server is notified.
- **Step 2:** This system informs the DataFeed of the users' current state of permissions, login status and Launch/Workstation IP address.
- **Step 3:** The DataFeed server will check the list of IP addresses sent to the API and if the Launch/Workstation IP is in the list, The DataFeed Server will also ensure the user is currently logged on, and pass information that the requirements were met. The Third Party system then has all the information it needs to permission their terminals.
- **Step 4:** If the user is not authenticated/logged into FactSet or the IP addresses do not match, then the third party system is not allowed to send exchange data to the third party terminal. If the user is authenticated/logged in and the IP addresses do match, then a second layer of permissioning takes place. The exchange data needs to be matched up with the permission map of the user by the third party server. If the end user has the proper permissions, then the exchange data can be displayed in the third party terminal, which runs on the same machine as the FactSet terminal.

If the end user's permission map does not contain the needed entitlement, a message will be sent saying the user is not entitled and no exchange data will be sent to the terminal.

Example:

1. User requests FDS-USA
2. An FDS-USA trade message containing permission code 12345 is returned by the DataFeed server
3. Third party confirms that user has 12345 in their permission map
4. Third party allows FDS-USA to be seen by user

Continuation of example:

5. The users's permission to 12345 is removed
6. FactSet provides notification of user's change in permission code to third party server
7. Third party server denies user access to FDS-USA

The login information, permission maps and IP address checks are dynamic. If there is any change, the third party server will be notified and the new logic should be applied.

5.3 Audit Process

FactSet has a number of tests designed to ensure the third party integrator is properly enforcing FactSet's permissions. These are contained in a separate document available upon request from FactSet. FactSet will need to perform an audit at the third party's office to ensure compliance.

5.4 Service and Data Model

The permission service name is FDS_PERM. The request keys to this service should be of the form USERNAME-SERIAL NUMBER (e.g., FDS-12345).

The permission request will return a response with 2 fields. FID 9221 (USER_LOGIN_STATUS), will return a 1 or 0. 1 signifies the client is logged in currently/authenticated and the IP addresses match. 0 signifies that the user is logged off/not authenticated or the IP addresses do not match.

For FID 9222 (USER_PERMISSIONS), there will be a comma delimited list of permission codes for the user. When a field exceeds 255 characters, the same 9222 fid is repeated with the new continuation of the permission code list. This continues until the list is complete.

The IP addresses need to be comma separated and sent through the method:

[Visual Basic 6.0]

```
Public Function request(service As String, key As String,
                        Optional snapshot_only As Boolean = False,
                        Optional options As String,
                        Optional ByRef is_pending As Boolean = False)
```

```
Set fdf = New FDF
Dim tag As Integer


tag = fdf.request("FDS_PERM", "USER-SERIAL", false, "1.2.3.4, 192.168.0.1");
```

[C#]

```
public int request(string service, string key, bool snapshot_only,
                   string options, out bool is_pending)
```

```
fdf = new FDFClass();

int tag = fdf.request("FDS_PERM", "USER-SERIAL", false, "1.2.3.4, 192.168.0.1",  out
pending);
```

Chapter 6 Options Greeks Calculation

FactSet provides additional fields that return Greeks values and Implied Volatilities for Streaming DataFeed users.

6.1 Requirements

The Options Greeks Calculations require Version 2.2 of the Exchange DataFeed COM Toolkit. Any applications that plan to use version 2.2 of the latest toolkit will need to recompile. Any applications that want to use this new functionality will require a code change and to recompile.

6.2 New Implied Volatility and Greek Fields[11]

Field Id	Name	Type	Description
2613	ANALYTIC_PRICE_RULE	Integer	This is a flag to tell which price is being used in the analytic calculations. A value of 1 means that Mid price is used. A value of 2 means that during market hours a Mid price will be used and after market hours the settlement price will be used.
2614	EXPIRATION_DAYS_TO	Integer	The number of business days until the option expires
2620	DELTA	Decimal	The rate of change of option value with respect to changes in the underlying asset's price.
2621	GAMMA	Decimal	The rate of change in the delta with respect to the changes in the underlying asset's price
2622	VEGA	Decimal	The sensitivity of the value of the option to the volatility of the underlying asset
2623	THETA	Decimal	The sensitivity of the value of the option to the passage of time
2624	RHO	Decimal	The sensitivity of the value of the option to the risk free interest rate
2630	IMP_VOL	Decimal	The volatility of the price of the underlying security that is implied by the market price of the option based on an option pricing model
2631	IMP_VOL_ASK	Decimal	The volatility of the price of the underlying security that is implied by the market ask price of the option based on an option pricing model
2632	IMP_VOL_BID	Decimal	The volatility of the price of the underlying security that is implied by the market bid price of the option based on an option pricing model
2633	IMP_VOL_CALC_RATE	Decimal	The calculated value of the interest rate using the option pricing model
2634	THEO_VALUE	Decimal	The calculated value of the option using the option pricing model

Please note that all fields except ANALYTIC_PRICE_RULE and EXPIRATION_DAYS_TO will be blank in the in the initial snapshot message. The values will begin streaming shortly after. The values will be recalculated based on any changes in the underlying asset or the option. The values will be sent at a maximum of once every 10 seconds.

[11] For detailed information on how these fields are calculated please see FactSet Online Assistant Page 14933

6.2.1 Sample Data

```
response key: IBM#A1814C195000-USA, tag: 1, msg:
T:1 K:IBM#A1814C195000-USA E:0 Flags:AGB
NumFids = 11 Size = 151
   MSG_TYPE[1] Val = U Size=1
   DELTA[2620] Val = 0.553972 Size=8
   GAMMA[2621] Val = 0.007738 Size=8
   VEGA[2622] Val = 0.901767 Size=8
   THETA[2623] Val = -0.020611 Size=9
   RHO[2624] Val = 1.200264 Size=8
   IMPL_VOL[2630] Val = 22.392416 Size=9
   IMPL_VOL_ASK[2631] Val = 22.697414 Size=9
   IMPL_VOL_BID[2632] Val = 22.087504 Size=9
   IMPL_VOL_CALC_RATE[2633] Val = 22.392416 Size=9
   THEO_VALUE[2634] Val = 20.424996 Size=9
```

6.3 Risk Free Interest Rates

FactSet uses Sovereign Debt Benchmarks for Risk Free Interest Rates. The country will be determined based on the currency of the option and the period of time will be determined based on the expiration date of the option.

6.4 Setting up Greek Calculations

To enable the Greek calculation the following step is required after the connection is made and before the option request is sent:

```
fdf.options_greeks = true;
```

Chapter 7 Level 2 Data

FactSet provides market depth in the Exchange DataFeed for Enterprise Streaming DataFeed users. The additional bid and ask information may be called Level 2, market depth, or order book data, depending on the exchange. In this document market depth is referred to as Level 2 data.

7.1 Requirements

Level 2 functionality in the COM toolkit requires version 2.5 or higher of the Exchange DataFeed COM Toolkit. Any applications that are updated to use version 2.5 of the latest toolkit will need to be recompiled. Any applications that are going to use level 2 functionality will require a code change and to recompile.

7.3 Setting up Level 2 Data

There are 2 ways to receive Level 2 data: Raw Data and Sorted Data. For either type, the ticker requested must be appended with ":L2", this will subscribe to the Level 2 feed for the given ticker and is the only requirement to subscribe to raw Level 2 data. To subscribe to NASDAQ TotalView data the ticker should be appended with :TV.

Additional FactSet product permissions are needed to consume these data sets. The raw data request will provide all the bids and asks for an individual security. The updates will be sent in the order they are received by FactSet. To access prerecorded canned data for development efforts use the service FDS_C, the available ticker for canned level 2 data is:

SIAC :
FDS
IBM
DIS
JNJ
WMT

NASDAQ :
CSCO
AAPL
INTC
MSFT
AMZN

Sorted Data Sorted Data is identical to raw data, with the exception that every valid Level 2 message contains an additional field indicating the message's sorted position (BID_INDEX_1 and ASK_INDEX_1). To enable Sorted Data, the Level 2 feature must be enabled after the connection is made before the Level 2 request is sent.

```
fdf.level2 = true;
```

This feature will allow consumers to create a display that has the bids/ask in price order.

7.2 Level 2 Fields

In addition to the currently supported data fields, the following table described the new fields added for the Level 2 content set.

Field Id	Name	Type	Description
150	BID_INDEX_1	Integer	Sorted Data only: The message's position in a sorted list of bids.
250	ASK_INDEX_1	Integer	Sorted Data only: The message's position in a sorted list of asks.
520	ORDER_CODE	String	Order Code
521	MM_STAT_BITMASK	String	Shows Open/Closed quotes[12]
522	MARKET_MECHANISM_TYPE	String	Used to show the order type as in market order or limit order
523	MARKET_MAKER_ID	String	Market Maker ID

Not every exchange will populate every new field that has been added. The new fields will be used with the level 1 fields BID_1/ASK_1, BID_VOL_1/ASK_VOL_1, and BID_TIME_1/ASK_TIME_1.

7.4 Processing Level 2 Data

There are a few specific rules for Level 2 messages that need to be followed to maintain an accurate record.

- If a message has the MSG_TYPE "D", it represents a delete, and the corresponding entry, by ORDER_CODE, is no longer valid. These must be processed properly to avoid stale date in the Level 2 record. For the Sorted Data functionality, it will no longer be considered when sorting the list and should be removed accordingly.
- For Sorted Data, each valid message will come with BID_INDEX_1 and/or ASK_INDEX_1 populated. These indicate the message's position in the sorted list of bids and ask respectively. To handle these messages properly, the previous corresponding entry in the list, by ORDER_CODE, should be removed, and this message should be inserted at the position specified in the INDEX field.

7.4.1 Processing a Message Example

The example code below shows one way to process a sorted Level 2 message from a callback. The callback function checks message type and bid/ask data, at which point any processing of that data can be done. In addition, it checks to see if the stream was closed, and if so, it closes the client-side stream by canceling the tag.

The server may close the stream at any time. In addition, error messages (like RT_E_NOT_FOUND) will cause the stream to set the close/end-of-stream indicator.

```
{
...
        fdf.level2 = true;
        bool pending;
        int tag = fdf.request("FDS1", "FDS-USA:L2", false, "", out pending);
}

private static void fdf_OnMessage(int tag, RT_Message msg)
{
        string msg_type  = msg.get((int)FIDS.MSG_TYPE);
        string bid_index = msg.get((int)FIDS.BID_INDEX_1);
        string ask_index = msg.get((int)FIDS.ASK_INDEX_1);
```

[12] Only used in Nasdaq Level 2 feed, this is the only level 2 exchange that does not clear the book at the end of the day, quotes are just closed.

```
        if (msg_type == "D") {
                // handle delete message
        }
        else {
                if (bid_index != "") {
                        // handle bid message
                }
                if (ask_index != "") {
                        // handle ask message
                }
        }
}
```

See the Level2Quote sample utility included in the C++ toolkit for a more complete example, including logic for maintaining sorted bid and ask lists.

See the Level2Com sample utility included in the COM toolkit for a more complete example, including logic for maintaining sorted bid and ask lists.

NOTE:The maximum number of simultaneous level 2 symbols per connection is limited to 100 symbols.

Appendix

Appendix A: A Complete C# Example

```
using System;
using FDSRTComLib; // for convenience
using System.Windows.Forms;

namespace SimpleExample
{
  class MainClass
  {
    private static void fdf_OnMessage(int tag, RT_Message msg)
    {
      if (msg.error)
      {
        Console.WriteLine("Error: {0}", msg.error_description);
        fdf.cancel(tag);
        return;
       }
       string msg_type = msg.get((int)FIDS.MSG_TYPE);
       string bid      = msg.get((int)FIDS.BID_1);
       string ask      = msg.get((int)FIDS.ASK_1);
       Console.WriteLine("Update: {0} Bid: {1} Ask: {2}", msg_type, bid, ask);

       // if the server closed the stream close our side as well
       if (msg.closed) { fdf.cancel(tag); }
    }
    [STAThread]
    static void Main(string[] args)
    {
      fdf = new FDFClass();
      fdf.OnMessage += new _IFDFEvents_OnMessageEventHandler(fdf_OnMessage);
      try
      {
        fdf.connection_info = "client:secret@api-stage.df.factset.com";
        fdf.connect(false);
      }
      catch(Exception e)
      {
        Console.WriteLine(e.Message);
        return;
      }

      bool pending;
      // create a real-time RT_Request for service=FDS1, symbol=FDS-USA
      int tag = fdf.request("FDS1", "FDS-USA", false, "",  out pending);
      Console.WriteLine("made a request for FDS-USA tag={0}", tag);
      // run a GUI event loop to dispatch our events
      Application.Run();
    }

    public static FDF fdf;
```

```
    }
}
```

Appendix B: Control Messages

The following table shows the possible control messages that can be delivered to the application-defined control callback procedure:

Control Type	Meaning	Additional Information
"DISCONNECTED"	The TCP Connection to the data server is disconnected.	The error information can be obtained via the errno property of the message object. The error will be one of the error values defined by the rt_errno enumeration.
"CONNECTED"	The TCP Connection to the data server is connected.	The current active service names are in the message. The FID, FIDS.SERVICE_NAME, is used repetitively.
"SERVICE_ENABLE"	New services are now available for requests.	The service names are in the message. The FID, FIDS.SERVICE_NAME, is used repetitively.
"SERVICE_DISCONNECT"	Services have become stale. Existing streams will now transition to stale.	The service names are in the message. The FID, FIDS.SERVICE_NAME, is used repetitively.
"SERVICE_DISABLE"	The services are no longer accepting any new streams.	The service names are in the message. The FID, FIDS.SERVICE_NAME, is used repetitively.
"TERMINATE"	The FactSet data server is requesting the application terminate its connection (i.e. the application MUST call disconnect())	This is typical when the authentication for an asynchronous connection has failed. Applications MUST call disconnect when receiving this message. If they do not, the API will call disconnect on its behalf.

The OnControl event handler has the following prototype:

[Visual Basic 6.0]
```
Private Sub fdf_OnControl(ByVal is_connected As Boolean,
                          ByVal msg As IRT_Message)
```

[C#]
```
private void fdf_OnControl(bool is_connected, RT_Message msg)
```

The is_connected boolean parameter will always indicate the current status of the TCP connection to the data server.

The control type is a string that can be extracted from the message key (e.g., control_type = msg.key). The enumeration of the possible control strings are listed in the table above.

Additional data fields may be present based on the type of control message. These fields are located in the RT_Message object.

Appendix C: Connection Strings and URI's

Connection Strings

Connection strings allow applications to specify host and authentication information as a single string. The syntax is as follows:

[USER][:PASSWD][@]***HOST***[:PORT][13]

The **HOST** value can be either a host name or dotted decimal (e.g., fdshost, api-stage.df.factset.com or 10.14.1.6). **HOST** is a mandatory parameter.

PORT can either be an integer or a service name (e.g., fdsserv or 6681). This is an optional parameter and defaults to 6681 if not specified.

The USER:PASSWD@ part is optional[14]. USER is the FactSet-supplied username, PASSWD is the FactSet-supplied password.

Examples:

client:aaa@fdshost means authenticate using a username of "client" and a password of "aaa" to the server "fdshost" on the default port 6681.

client@10.2.4.5:4063 means connect to the host at 10.2.4.5 on port 4063 with a username set to client.

Connection URI's

Connection URI's are universal resource identitifiers that allow the API to resolve a connection string and its individual components. If a specific protocol is not given, the URI itself is a connection string.

An application may request the API to look for connection information in a file or a Windows registry. In this case, the application should pass in a valid configuration URI (see Appendix D). If an empty string is passed in as the URI, FDF.get_property() will be used to resolve the connection information. The following table outlines the property names used for resolution:

Property Name	Meaning
RT_CONNECTION	Actual Connection string of the form: [USER][:PASSWD][@]***HOST***[:PORT]
RT_HOST	The hostname of the Data Server format: **HOST**[:PORT]
RT_USER	Username for authentication
RT_PASSWORD	Password for authentication

The RT_CONNECTION parameter is queried first followed by RT_HOST, RT_USER, and RT_PASSWORD. If present, the RT_HOST, RT_USER, and RT_PASSWORD will overide any specific values obtained from the RT_CONNECTION parameter.

[13] [ ] indicate optional
[14] Although the user and password information is optional in the connection string, the API must have a user and password in order to authenticate with the FactSet Data Server.

Appendix D: Configuration Properties

The API supports loading global properties from a file or from the Windows Registry. This can be done using the global method: ***FDF.load_properties.***

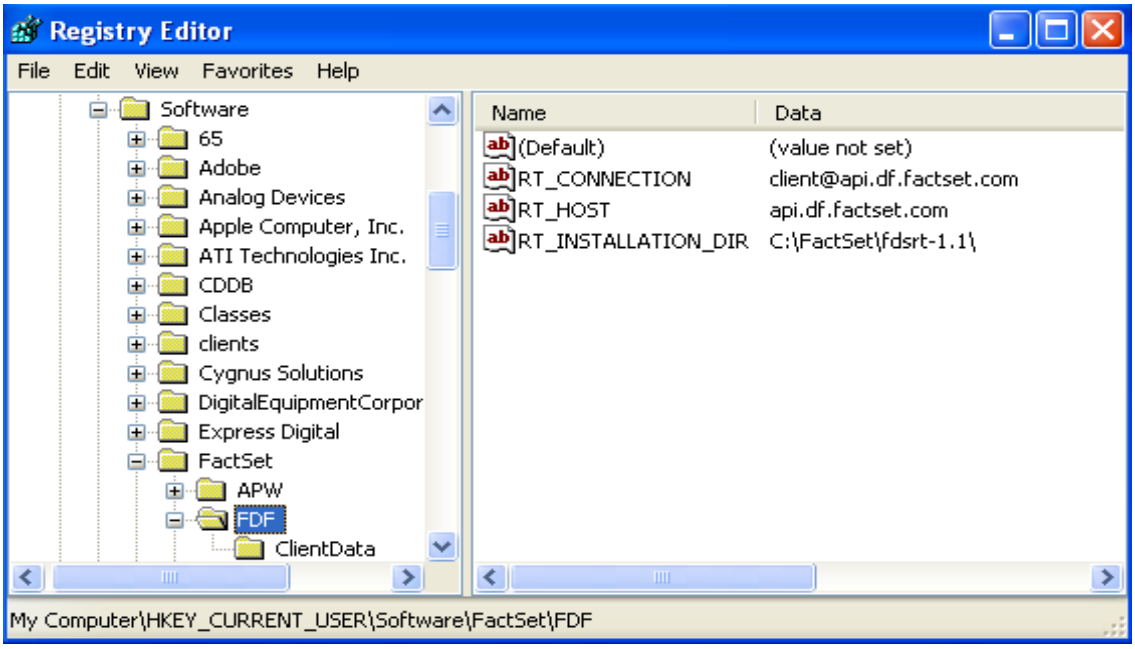
[Visual Basic 6.0]
```
Public Sub load_properties(uri As String, Optional append As Boolean = False)
```

[C#]
```
public void load_properties(string uri, bool append)
```

Loading Properties from the Windows Registry

- **FDF.load_properties("reg:/HKEY_LOCAL_MACHINE/Software/FactSet/FDF")** – opens the registry hive HKEY_LOCAL_MACHINE. The method will also traverse to the *Software/FactSet/FDF* section and recursively load all the name/value pairs.
- **FDF.load_properties("reg:/HKEY_LOCAL_MACHINE|HKEY_CURRENT_USER/Software/FactSet/FDF")** – This method performs the same action as the previous example. In addition, after traversing the HKEY_LOCAL_MACHINE hive, the HKEY_CURRENT_USER hive will be read. Any property names that are duplicated in the HKEY_CURRENT_USER hive will overwrite the values read in from the HKEY_LOCAL_MACHINE hive.

Example registry hive:



Based on the example above the following properties will be loaded:

Property Name	Value
RT_CONNECTION	client@api.df.factset.com
RT_HOST	api.df.factset.com
RT_INSTALLATION_DIR	C:\FactSet\fdsrt-2\
ClientData/Property1[15]	value1

Loading Properties from a File

By default, the uri string without a specific protocol is a file. Some examples:

- **FDF.load_properties("etc\my_config.txt")** – opens "my_config.txt" using the relative path "etc\".
- **FDF.load_properties("file:etc\my_config.txt")** – same as the previous example

An example configuration file:

```
//
// C++ style comments are used
//
RT_CONNECTION         = "client@fdshost:6690";
FIELD_MAP_FILE        = "etc\rt_fields.xml";
BASEDIR               = "db\";
SYMBOL_FILE           = "etc\USE_tickers.txt";
ClientData::Property1 = "value1"
```

Each name/value pair must end in a semi-colon. The syntax is described as follows:

{parameter_name} = "value in quotations";

Syntax rules:

- Anything between the '//' character sequence and the new-line character is a comment.
- Parameter names must be a single word (whitespace is not permitted).
- Parameter names and values must be separated by the '=' character.
- All parameter values must be in quotations, and end with a ';' character.
- Any amount of whitespace is permitted on either side of the '=' delimiter.
- Values must be in quotations and should not contain the new line character.

Based on the example above the following properties will be loaded:

Property Name	Value

[15] If additional subkeys exist within the given registry path (like ClientData in the above example), the property name will include the key followed by a '/'. In the above example all values in the ClientData section will have a property name ClientData/{name}".

RT_CONNECTION	client@fdshost:6690
FIELD_MAP_FILE	etc\rt_fields.xml
BASEDIR	db\
SYMBOL_FILE	etc\USE_tickers.txt
ClientData::Property1	value1

Appendix E: Document Version History

The following are revisions made since the Version 1.0 revision A

Revisions	Sections
Added documentation for workstation connection	1-4
Added progid's for COM Classes	2.2.3
Added Required Ports	1.4.2

The following are revisions made since the Version 1.2 revision D

Revisions	Sections
Added Chapter on Greeks	6
Added Level 2 data	7
Added details on OTP support	3

The following are revisions made since the Version 3.0 revision A

Revisions	Sections
Added Chapter on Greeks	6