Enterprise Transport API C# Edition 3.2.0.L1

TRAINING EXAMPLE OVERVIEW

Document Version: 3.2.0
Date of issue: April 2024
Document ID: ETACSharp320CT.240

REFINITIV™

Contents

1 Training Examples Suite

1.1 Overview

The Enterprise Transport API training example suite provides customers with step-by-step training for consumer, interactive provider, and non-interactive provider applications. An example is provided for each application type (consumer, interactive provider, and non-interactive provider), illustrating how to build an application from the ground up. Each example starts at a very basic level and adds functionality with each step. The examples are meant to simulate a classroom environment starting with basic concepts and building on those concepts resulting in a basic functional application.

1.2 Audience

The Enterprise Transport API training example suite is designed to aid programmers using the Enterprise Transport API. This manual is written for readers who are members of programming staff involved in the design, coding, and test phases for applications that use the Enterprise Transport API. The reader should be familiar with the data types, operational characteristics, and user requirements of real-time data delivery networks, and be experienced in developing products using the C# programming language in a networked environment.

1.3 Conventions

1.3.1 Typographic

- Functions, arguments, and data structures are shown in **`orange, Courier New`** font.
- Parameters, filenames, tools, utilities, and directories are shown in **Bold** font.
- Document titles and variable values are shown in *italics*.
- When initially introduced, concepts are shown in ***Bold, Italics***.

1.3.2 Programming

Enterprise Transport API C# Edition Standard conventions were followed.

2 Consumer Training Example

2.1 Overview

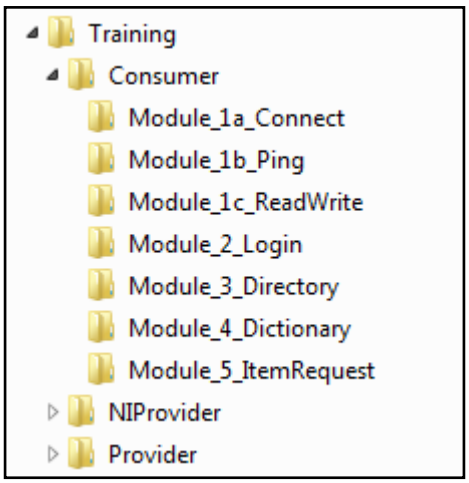
The ***Consumer Training*** example (also called **Consumer**) provides detailed step-by-step code modules that show how to create a simple, basic Open Message Model consumer application using the Enterprise Transport API.

The **Consumer** example application:

- Provides an implementation of an Open Message Model consumer application.
- Consists of several modules that show incremental progress of how to build a basic consuming application.
- Is written with simplicity in mind and demonstrates the basic Enterprise Transport API functionality.

You can use file comparison tools (such as **WinMerge**, etc.) to compare any two consecutive modules. This can help you understand new code added to any specific module. You can also compare each module of the **Consumer** training example code with the same module name (if it exists) from another training application, such as the **NIProvider** training example to find commonality between the applications.

The application is installed in the **Training\Consumer** folder and has seven available modules as illustrated:



2.2 Consumer Training Module 1a: Establish Network Communication

The first step of any Enterprise Transport API consumer application is to establish a network connection with its peer component (i.e., another application with which to interact). An Open Message Model consumer typically creates an outbound connection to the well-known hostname and port of a server (i.e., the interactive provider or Refinitiv Real-Time Advanced Distribution Server). The consumer uses the `Connect()` function to initiate the connection and then uses the `InitChannel()` function to complete channel initialization.

2.3 Consumer Training Module 1b: Ping (heartbeat) Management

Ping or heartbeat messages indicate the continued presence of an application. After the consumer's connection is active, ping messages must be exchanged. The negotiated ping timeout is retrieved using the `GetChannelInfo()` function. The connection will be terminated if ping heartbeats are not sent or received within the expected time frame.

2.4 Consumer Training Module 1c: Handling Reads and Writes

When channel initialization is complete, the state of the channel (`lChannel.State`) is **ChannelState.ACTIVE**, and applications can send and receive data.

2.5 Consumer Training Module 2: Perform Login Process

Applications authenticate using the Login domain model. An Open Message Model consumer must authenticate with a provider using a login request prior to issuing any other requests or opening any other streams. After receiving a login request, an interactive provider determines whether a user is permissioned to access the system. The interactive provider sends back a Login response, indicating to the consumer whether access is granted.

2.6 Consumer Training Module 3: Obtaining Source Directory

The Source Directory domain model conveys information about all available services in the system. An Open Message Model consumer typically requests a Source Directory to retrieve information about available services and their capabilities. This includes information about supported domain types, the service's state, the quality of service (QoS), and any item group information associated with the service.

2.7 Consumer Training Module 4: Load or Download Dictionary

Consumer applications often require a dictionary for encoding or decoding specific pieces of information. This dictionary typically defines type and formatting information. Content that uses the **`FieldList`** type requires the use of a field dictionary (usually the Refinitiv **RDMFieldDictionary**, although it could also be a user-defined or user-modified field dictionary).

A consumer application can choose whether to load necessary dictionary information from a local file or download the information from an available provider.

2.8 Consumer Training Module 5: Issue Item Requests

After the consumer application successfully logs in and obtains Source Directory and Dictionary information, it can request additional content. When issuing the request, the consuming application specifies the **`serviceId`** of the desired service along with a ***streamId***. Requests can be sent for any domain using the formats defined in that domain model specification. In this simple example, we show how to make a Market Price level I data Item request to obtain the data from a provider.

3 Interactive Provider Training Example

3.1 Overview

The ***Interactive Provider Training*** example (also called **Provider**) provides detailed step-by-step code modules on how to create an Open Message Model interactive provider application.

The **Provider** example application:

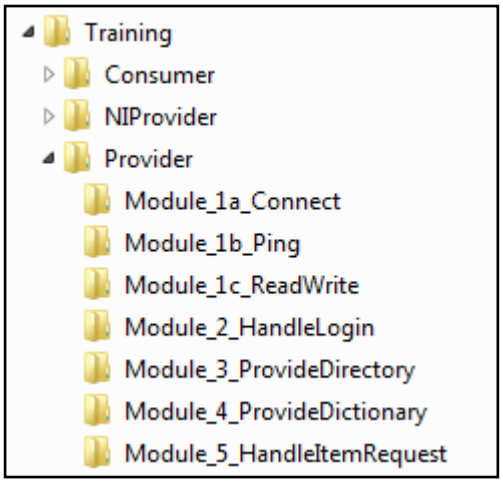
- Provides an implementation of an Open Message Model interactive provider application.
- Is written with simplicity in mind and demonstrates the basic functionality of the Enterprise Transport API.
- Consists of several modules that show incremental progress showing how to build the Enterprise Transport API Interactive Provider application.

You can use file comparison tools (such as **WinMerge**, etc.) to compare any two consecutive modules. This can help you understand any new code added to any specific module

You can also compare each module of the Interactive Provider Training example code with the same module name (if it exists) of another training application (such as the ***Enterprise Transport API NIProvider Training*** example) to find commonality between the applications.

Content is encoded and decoded using the Enterprise Transport API Message Package and the Enterprise Transport API Data Package.

The application is installed in the **Training\Provider** folder and has seven available modules:



3.2 Interactive Provider Training Module 1a: Establish Network Communication

The first step of any Enterprise Transport API interactive provider application is to establish a listening socket on a well-known port so that consumer applications can easily connect. The provider uses the **`Bind()`** method to open the port and listen for incoming connection attempts.

Whenever an Open Message Model consumer application attempts to connect, the provider uses the **`Accept()`** method to begin the connection initialization process.

For this simple training application, the interactive provider only supports a single client.

3.3 Interactive Provider Training Module 1b: Ping (heartbeat) Management

Ping or heartbeat messages indicate the continued presence of an application. After establishing a connection, ping messages must be exchanged. The negotiated ping timeout is retrieved using the **`GetChannelInfo()`** method. If ping heartbeats are not sent or received within the expected time frame, the connection is terminated.

3.4 Interactive Provider Training Module 1c: Handling Reads and Writes

When channel initialization is complete, the state of the channel (**`lChannel.State`**) is **ChannelState.ACTIVE** and applications can send and receive data

3.5 Interactive Provider Training Module 2: Perform Login Process

Applications authenticate using the Login domain model. An Open Message Model interactive provider must handle the consumer's Login request messages and supply appropriate responses. After receiving a Login request, the interactive provider must perform authentication and permissioning.

3.6 Interactive Provider Training Module 3: Providing Source Directory

An Open Message Model provider provides source directory information specifying available services and their capabilities. This includes information about supported domain types, the service's state, the quality of service (QoS), and any item group information associated with the service.

3.7 Interactive Provider Training Module 4: Providing Dictionary

Applications often require the use of a dictionary for encoding or decoding specific pieces of information. This dictionary typically defines type and formatting information. Content that uses the **`FieldList`** type requires the use of a field dictionary (usually the Refinitiv **RDMFieldDictionary**, though it could also be a user-defined or user-modified field dictionary).

Providers may optionally provide dictionary information to consumers.

3.8 Interactive Provider Training Module 5: Item Requests

Interactive providers must be able to accept item requests from a consumer and provide an appropriate response. In this simple example, we show how to handle a Market Price level I data Item request and send a response.

4 Non-Interactive Provider Training Example

4.1 Overview

The ***Non-Interactive Provider Training*** example (also called **NIProvider**) provides detailed step-by-step code modules that show how to create an Open Message Model non-interactive provider application.

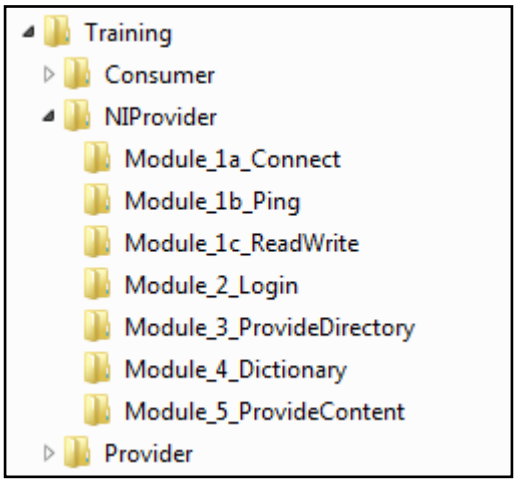
The **NIProvider** example application:

- Provides an implementation of an Open Message Model non-interactive provider application.
- Consists of several modules that show incremental progress showing how to build the Enterprise Transport API Non-Interactive Provider application.
- Is written with simplicity in mind and demonstrates the basic functionality of the Enterprise Transport API.
- Is written such that you can use file comparison tools (such as **WinMerge**, etc.) to compare any two consecutive modules. This can help you understand any new code added to a specific module.

You can also compare each module of the Non-Interactive Provider Training example code with the same module name (if one exists) of another training application (e.g., the Enterprise Transport API Interactive Provider Training example) to find commonality between the Enterprise Transport API Interactive Provider Training and Enterprise Transport API Non-Interactive Provider Training applications, if you wish.

Content is encoded and decoded using the Enterprise Transport API Message Package and the Enterprise Transport API Data Package.

The application is installed in the **Training\NIProvider** folder and has seven available modules:



4.2 Non-Interactive Provider Training Module 1a: Establish Network Communication

The first step of any Enterprise Transport API non-interactive provider application is to establish network communication with a Refinitiv Real-Time Advanced Data Hub. To do so, the Open Message Model non-interactive provider creates an outbound connection to the well-known hostname and port of a Refinitiv Real-Time Advanced Data Hub. The non-interactive provider uses the **`Connect()`** method to initiate the connection process and uses the **`InitChannel()`** method to complete channel initialization.

4.3 Non-Interactive Provider Training Module 1b: Ping (heartbeat) Management

Ping or heartbeat messages indicate the continued presence of an application. After establishing a connection, ping messages must be exchanged. The negotiated ping timeout is retrieved using the **`GetChannelInfo()`** method. If ping heartbeats are not sent or received within the expected time frame, the connection will be terminated.

4.4 Non-Interactive Provider Training Module 1c: Handling Reads and Writes

When channel initialization is complete, the state of the channel (**`lChannel.State`**) is **ChannelkState.ACTIVE** and it is possible for an application to send or receive data.

4.5 Non-Interactive Provider Training Module 2: Perform Login Process

Applications authenticate using the Login domain model. An Open Message Model non-interactive provider must authenticate with the Refinitiv Real-Time Advanced Data Hub using a Login request prior to providing any content. After receiving a Login request, a Refinitiv Real-Time Advanced Data Hub determines whether a user is permissioned to access the system. The Refinitiv Real-Time Advanced Data Hub sends a Login response back to the Non-Interactive Provider indicating whether access is granted.

4.6 Non-Interactive Provider Training Module 3: Providing Source Directory

The Source Directory domain model conveys information about all available services in the system and the Open Message Model non-interactive provider application must provide Source Directory information specifying all available services that it provides.

This includes information about supported domain types, the service's state, the quality of service (QoS), and any item group information associated with the service.

4.7 Non-Interactive Provider Training Module 4: Loading Dictionary

Some data requires the use of a dictionary for encoding or decoding specific pieces of information. This dictionary typically defines type and formatting information. Content that uses the **`FieldList`** type requires the use of a field dictionary (usually the Refinitiv **RDMFieldDictionary**, though it could also be a user-defined or modified field dictionary).

The Open Message Model non-interactive provider will use dictionaries that are available locally in a file.

4.8 Non-Interactive Provider Training Module 5: Providing Content

After providing a Source Directory, the Open Message Model non-interactive provider application can begin pushing content to the Refinitiv Real-Time Advanced Data Hub. In this simple example, we show functions for sending one Market Price domain Item refresh, update, and close status message(s) to a Refinitiv Real-Time Advanced Data Hub.

Document Version: 3.2.0
Date of issue: April 2024
Document ID: ETACSharp320CT.240