ETP DevKit – Getting Started

Version 1.1

Copyright 2016 Energistics

Contents

1 Introduction

This guide introduces the ETP DevKit library and gives examples for both client and server scenarios. The source code for this library and all of the samples used in this document can be found in the following Git repository:

https://bitbucket.org/energistics/etp-devkit

The ETP DevKit was developed by Petrotechnical Data Systems (PDS) and contributed to Energistics. It is provided as an open source project under the Apache License, Version 2.0. Further development will be guided by Energistics and the user community.

1.1 Intended Audience

Solution architects evaluating Energistics Transfer Protocol technologies and software developers writing .NET applications which are required to send and receive asynchronous ETP messages. This guide assumes familiarity with the ETP Specification, which can be downloaded from Energistics.

1.2 Prerequisites

This document assumes the following development environment

- Windows Server 2012+ or Windows 8+
- Visual Studio 2013 or 2015
- .NET Framework 4.5.2

Additionally, for native Web Socket support and ASP.NET

- IIS 8.5+ with Web Sockets enabled

2 Overview

The ETP DevKit is a .NET library providing a common foundation and the basic infrastructure needed to communicate via the Energistics Transfer Protocol. The library provides a definition and base implementation of each interface described in the ETP Specification.

Each interface implementation has been developed as a protocol handler that can be used out of the box or extended to provide additional functionality. Customized processing of messages can be achieved by either registering handlers for the various interface events or by deriving from the library's protocol handlers and overriding the virtual message handling methods.

The base handlers can be bypassed altogether by implementing and registering a custom implementation of the protocol specific interfaces.

2.1 Class Diagrams

The following class diagrams provide an overview of the types found in the ETP DevKit library.

2.1.1 Base Types

2.1.2   Protocol Handlers

ICoreClient
Interface
→ IProtocolHandler

Methods
- CloseSession
- RequestSession

Events
- OnCloseSession
- OnOpenSession

ICoreClient

CoreClientHandler
Class
→ EtpProtocolHandler

Properties
- ServerObjects
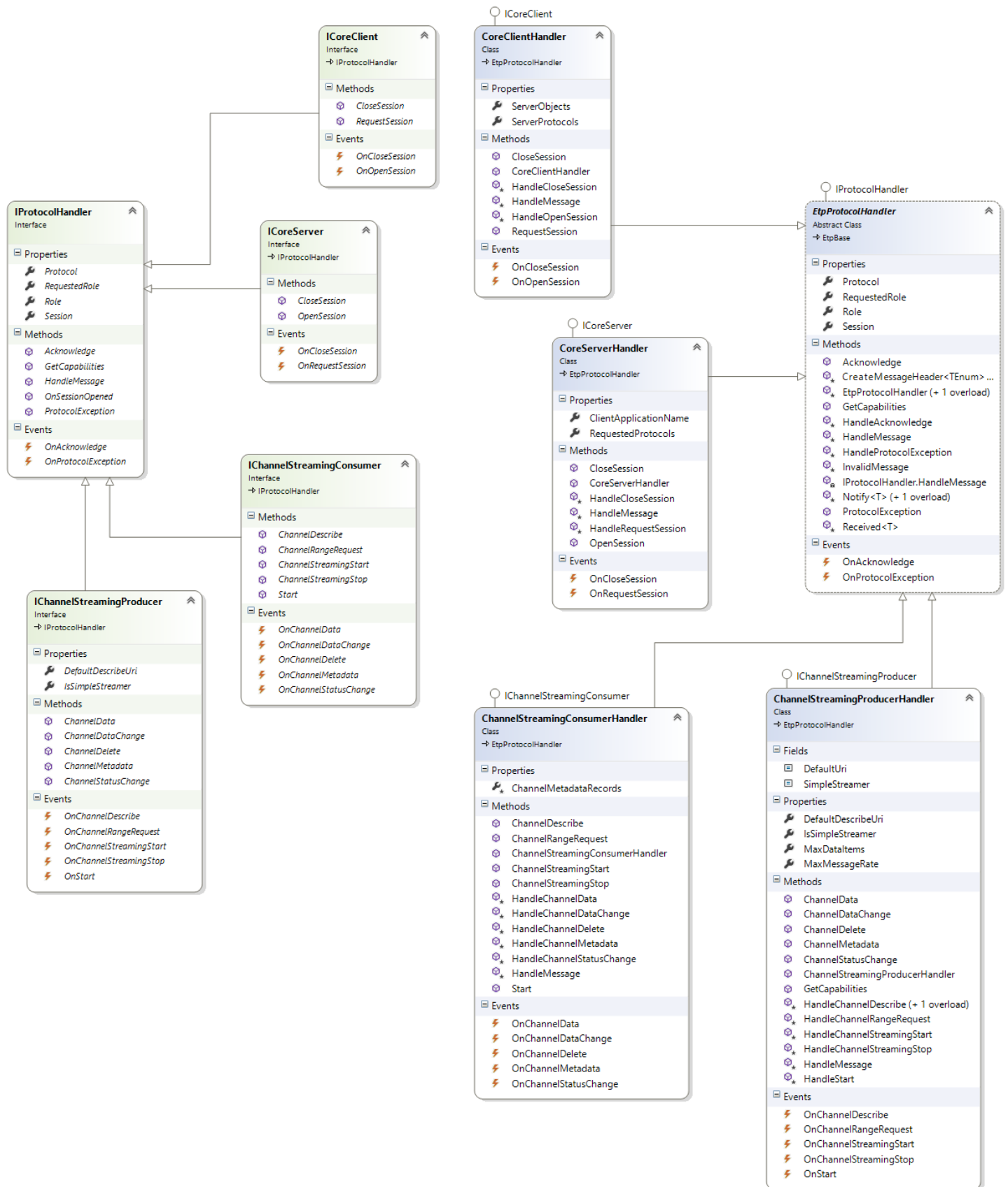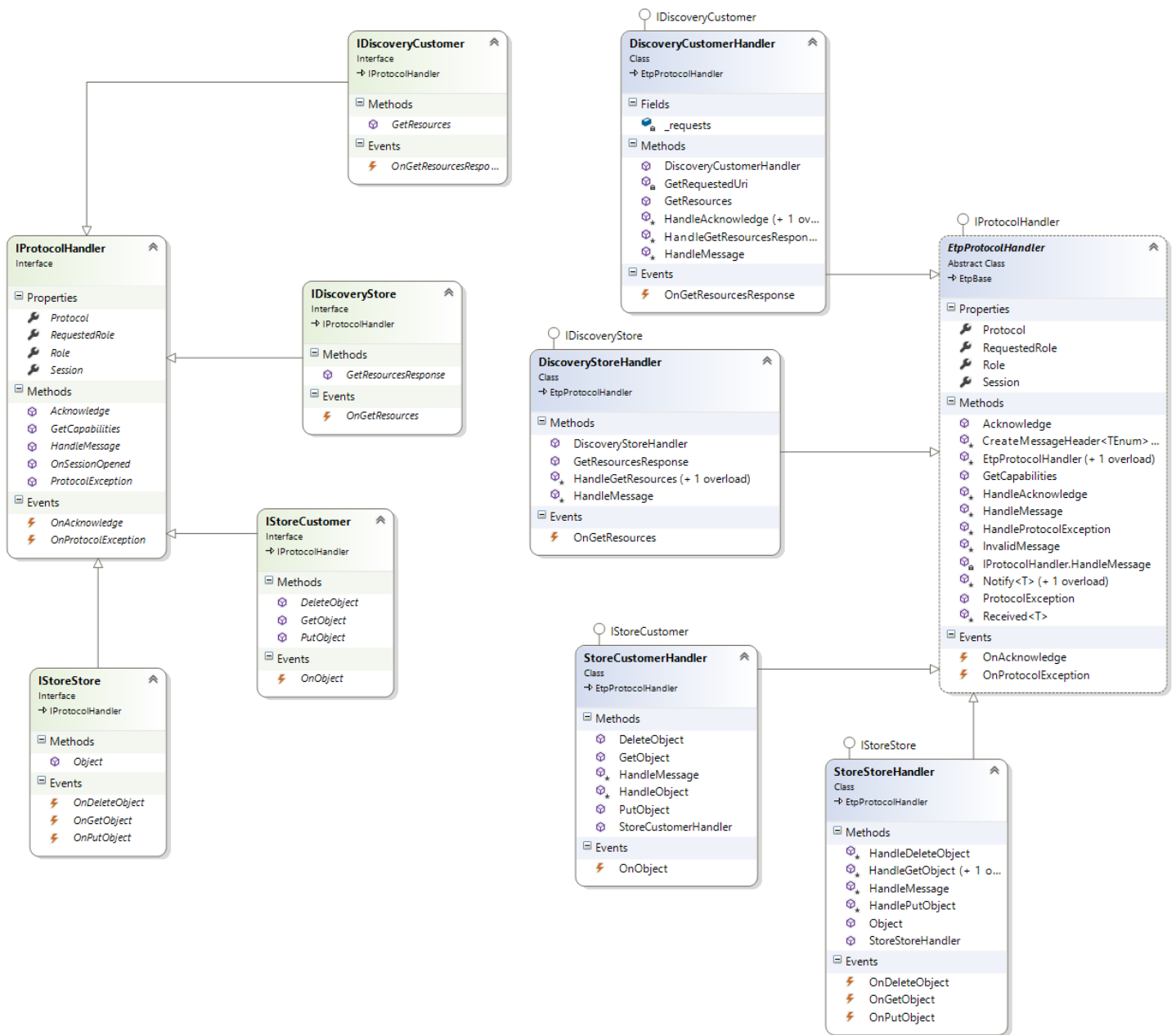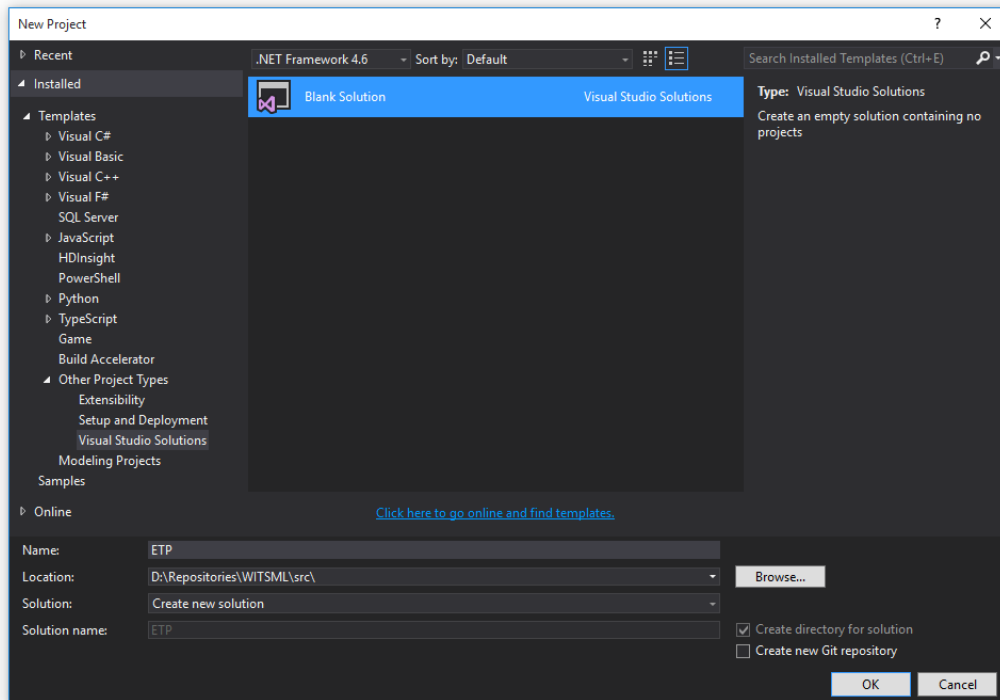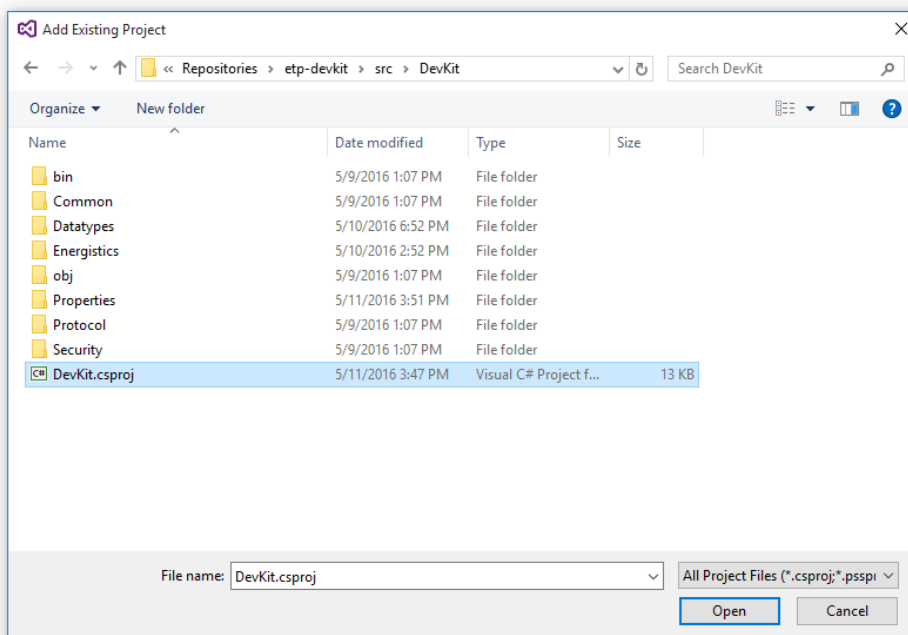- ServerProtocols

Methods
- CloseSession
- CoreClientHandler
- HandleCloseSession
- HandleMessage
- HandleOpenSession
- RequestSession

Events
- OnCloseSession
- OnOpenSession

IProtocolHandler
Interface

Properties
- Protocol
- RequestedRole
- Role
- Session

Methods
- Acknowledge
- GetCapabilities
- HandleMessage
- OnSessionOpened
- ProtocolException

Events
- OnAcknowledge
- OnProtocolException

ICoreServer
Interface
→ IProtocolHandler

Methods
- CloseSession
- OpenSession

Events
- OnCloseSession
- OnRequestSession

IProtocolHandler

EtpProtocolHandler
Abstract Class
→ EtpBase

Properties
- Protocol
- RequestedRole
- Role
- Session

Methods
- Acknowledge
- CreateMessageHeader<TEnum> ...
- EtpProtocolHandler (+ 1 overload)
- GetCapabilities
- HandleAcknowledge
- HandleMessage
- HandleProtocolException
- InvalidMessage
- IProtocolHandler.HandleMessage
- Notify<T> (+ 1 overload)
- ProtocolException
- Received<T>

Events
- OnAcknowledge
- OnProtocolException

ICoreServer

CoreServerHandler
Class
→ EtpProtocolHandler

Properties
- ClientApplicationName
- RequestedProtocols

Methods
- CloseSession
- CoreServerHandler
- HandleCloseSession
- HandleMessage
- HandleRequestSession
- OpenSession

Events
- OnCloseSession
- OnRequestSession

IChannelStreamingConsumer
Interface
→ IProtocolHandler

Methods
- ChannelDescribe
- ChannelRangeRequest
- ChannelStreamingStart
- ChannelStreamingStop
- Start

Events
- OnChannelData
- OnChannelDataChange
- OnChannelDelete
- OnChannelMetadata
- OnChannelStatusChange

IChannelStreamingProducer
Interface
→ IProtocolHandler

Properties
- DefaultDescribeUri
- IsSimpleStreamer

Methods
- ChannelData
- ChannelDataChange
- ChannelDelete
- ChannelMetadata
- ChannelStatusChange

Events
- OnChannelDescribe
- OnChannelRangeRequest
- OnChannelStreamingStart
- OnChannelStreamingStop
- OnStart

IChannelStreamingConsumer

ChannelStreamingConsumerHandler
Class
→ EtpProtocolHandler

Properties
- ChannelMetadataRecords

Methods
- ChannelDescribe
- ChannelRangeRequest
- ChannelStreamingConsumerHandler
- ChannelStreamingStart
- ChannelStreamingStop
- HandleChannelData
- HandleChannelDataChange
- HandleChannelDelete
- HandleChannelMetadata
- HandleChannelStatusChange
- HandleMessage
- Start

Events
- OnChannelData
- OnChannelDataChange
- OnChannelDelete
- OnChannelMetadata
- OnChannelStatusChange

IChannelStreamingProducer

ChannelStreamingProducerHandler
Class
→ EtpProtocolHandler

Fields
- DefaultUri
- SimpleStreamer

Properties
- DefaultDescribeUri
- IsSimpleStreamer
- MaxDataItems
- MaxMessageRate

Methods
- ChannelData
- ChannelDataChange
- ChannelDelete
- ChannelMetadata
- ChannelStatusChange
- ChannelStreamingProducerHandler
- GetCapabilities
- HandleChannelDescribe (+ 1 overload)
- HandleChannelRangeRequest
- HandleChannelStreamingStart
- HandleChannelStreamingStop
- HandleMessage
- HandleStart

Events
- OnChannelDescribe
- OnChannelRangeRequest
- OnChannelStreamingStart
- OnChannelStreamingStop
- OnStart

[4]

IDiscoveryCustomer
Interface
→ IProtocolHandler

Methods
- GetResources

Events
- OnGetResourcesRespo...

○ IDiscoveryCustomer

DiscoveryCustomerHandler
Class
→ EtpProtocolHandler

Fields
- _requests

Methods
- DiscoveryCustomerHandler
- GetRequestedUri
- GetResources
- HandleAcknowledge (+ 1 ov...
- HandleGetResourcesRespon...
- HandleMessage

Events
- OnGetResourcesResponse

○ IProtocolHandler

EtpProtocolHandler
Abstract Class
→ EtpBase

Properties
- Protocol
- RequestedRole
- Role
- Session

Methods
- Acknowledge
- CreateMessageHeader<TEnum> ...
- EtpProtocolHandler (+ 1 overload)
- GetCapabilities
- HandleAcknowledge
- HandleMessage
- HandleProtocolException
- InvalidMessage
- IProtocolHandler.HandleMessage
- Notify<T> (+ 1 overload)
- ProtocolException
- Received<T>

Events
- OnAcknowledge
- OnProtocolException

IProtocolHandler
Interface

Properties
- Protocol
- RequestedRole
- Role
- Session

Methods
- Acknowledge
- GetCapabilities
- HandleMessage
- OnSessionOpened
- ProtocolException

Events
- OnAcknowledge
- OnProtocolException

IDiscoveryStore
Interface
→ IProtocolHandler

Methods
- GetResourcesResponse

Events
- OnGetResources

○ IDiscoveryStore

DiscoveryStoreHandler
Class
→ EtpProtocolHandler

Methods
- DiscoveryStoreHandler
- GetResourcesResponse
- HandleGetResources (+ 1 overload)
- HandleMessage

Events
- OnGetResources

IStoreCustomer
Interface
→ IProtocolHandler

Methods
- DeleteObject
- GetObject
- PutObject

Events
- OnObject

○ IStoreCustomer

StoreCustomerHandler
Class
→ EtpProtocolHandler

Methods
- DeleteObject
- GetObject
- HandleMessage
- HandleObject
- PutObject
- StoreCustomerHandler

Events
- OnObject

IStoreStore
Interface
→ IProtocolHandler

Methods
- Object

Events
- OnDeleteObject
- OnGetObject
- OnPutObject

○ IStoreStore

StoreStoreHandler
Class
→ EtpProtocolHandler

Methods
- HandleDeleteObject
- HandleGetObject (+ 1 o...
- HandleMessage
- HandlePutObject
- Object
- StoreStoreHandler

Events
- OnDeleteObject
- OnGetObject
- OnPutObject

[5]

3   Solution and Project Setup

3.1   New Solution

Open Visual Studio and create a new, blank solution named ETP.

3.2   Import ETP DevKit

Add an existing project to the solution by browsing to the DevKit.csproj file located in the project downloaded from the Git repository.

3.3 New Project

Create a new Unit Test project named ETP.Tests
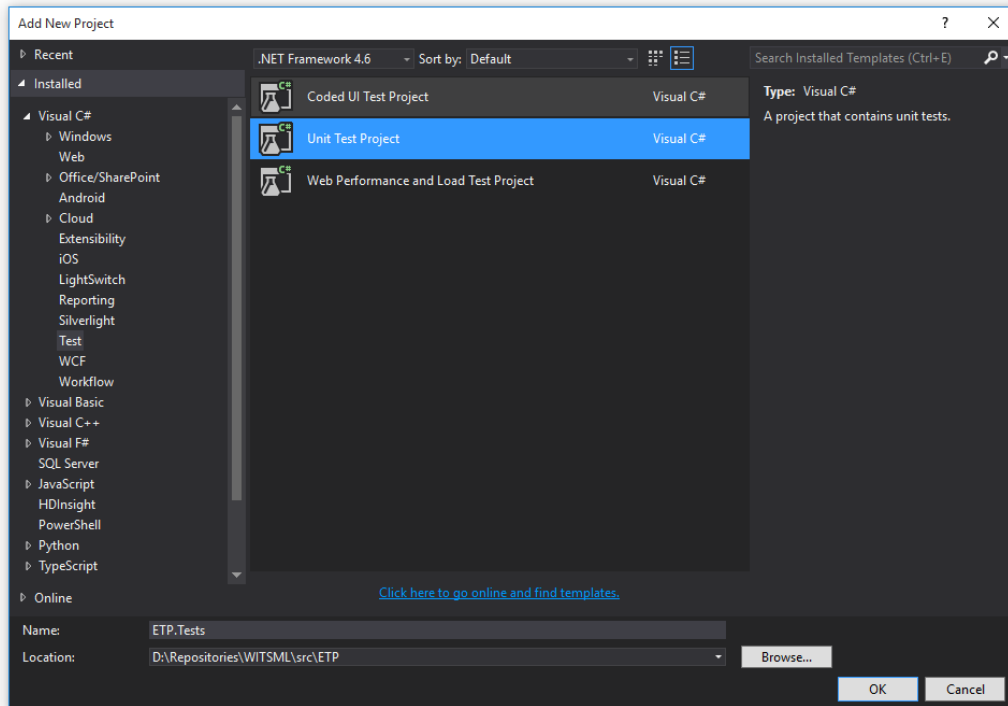
Rename the UnitTest1.cs file to EtpClientTests.cs.

Open the project properties for ETP.Tests and select ".NET Framework 4.5.2" as the target framework.
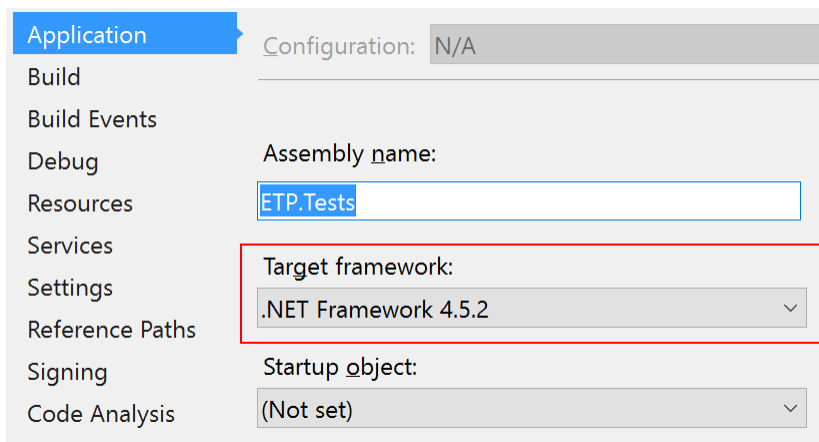
3.4 Update Dependencies

Open the NuGet Package Manager Console (Tools --> NuGet Package Manager) and click Restore to download any missing NuGet packages.

Add a project reference to the ETP.Tests project for the DevKit library.
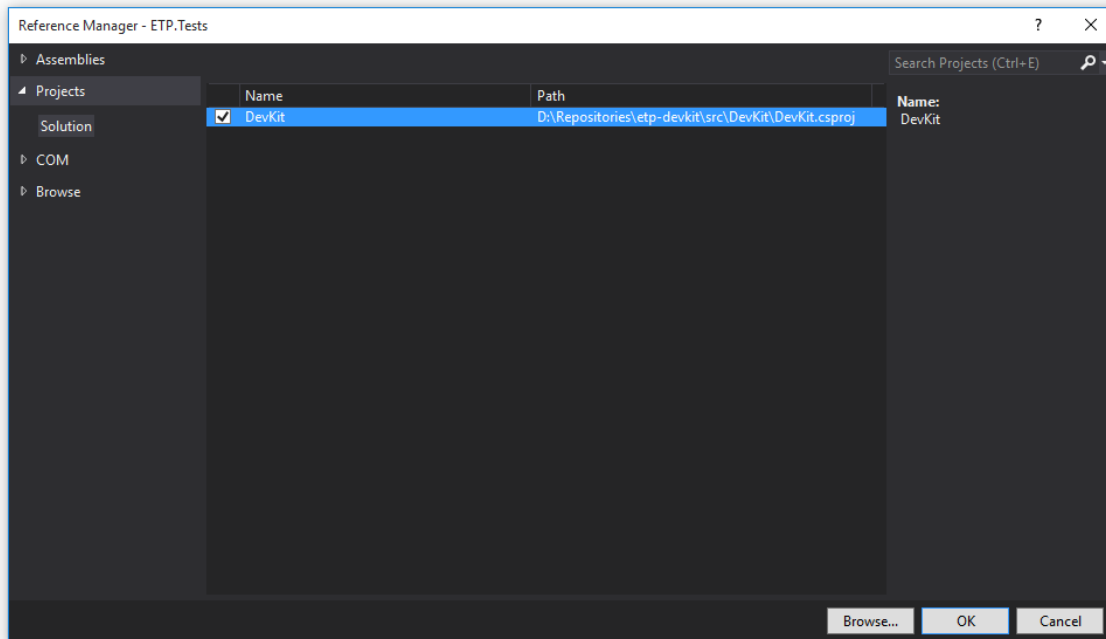
Add the following dependencies to the ETP.Tests project using the NuGet Package Manager Console or via the Manage NuGet Packages dialog:

- log4net (version 2.0.5)
- Newtonsoft.Json (version (9.0.1+)
- Apache.Avro (version 1.7.7.2)
- ETP (version 1.5.0) – ETP messages library
- WebSocket4Net (version 0.14.1) – Web Socket Client library
- SuperWebSocket (version 0.9.0.2) – Web Socket Server library

When completed, your environment should look similar to the following screen shot.
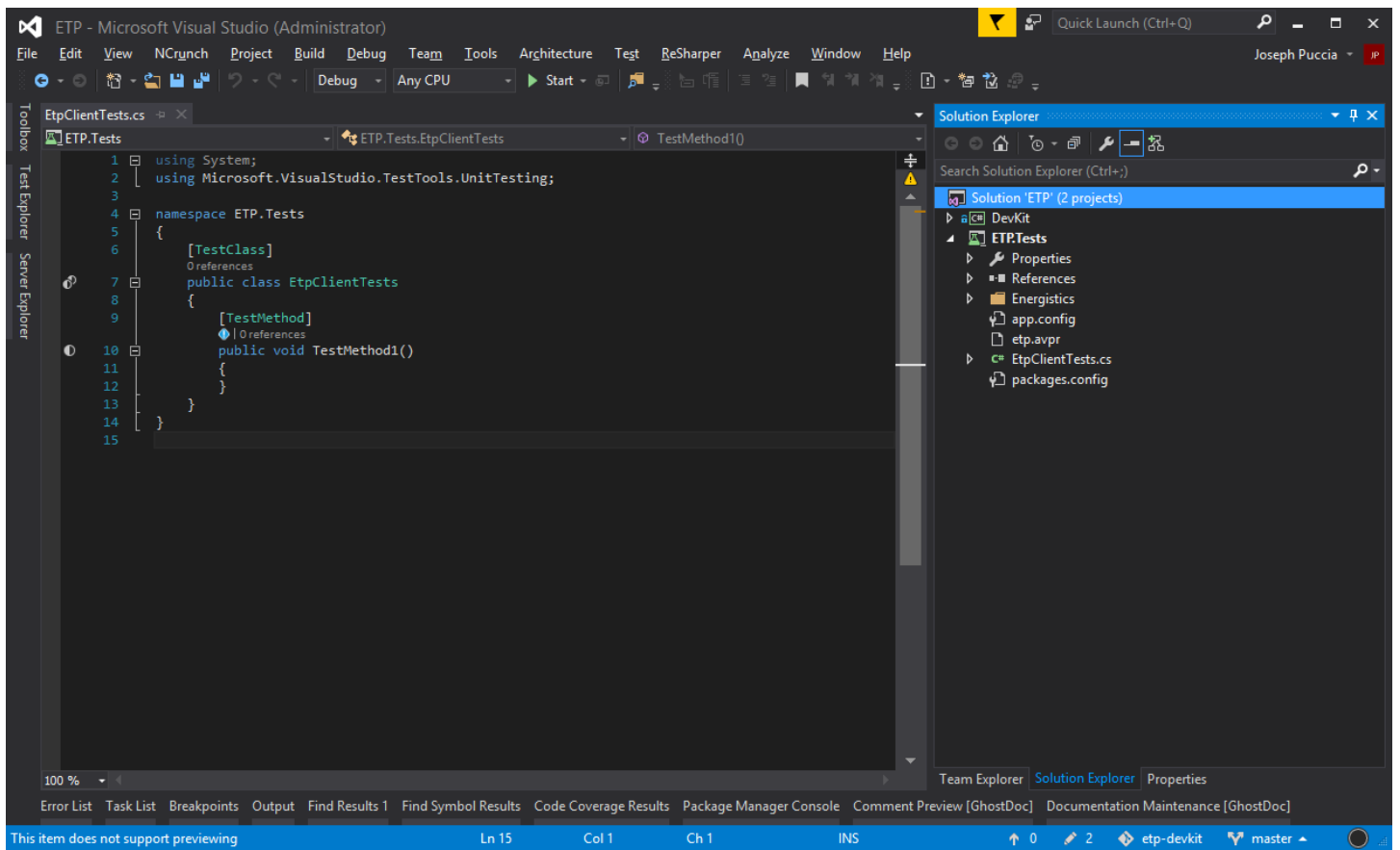
4 Integration Tests

4.1 Open Web Socket Connection

Replace the entire contents of the EtpClientTests.cs file with the following, or rename the default method, TestMethod1, to EtpClient_Opens_WebSocket_Connection and copy-and-paste the following code snippet.

```csharp
using System.Threading.Tasks;
using Energistics.Protocol.ChannelStreaming;
using Energistics.Protocol.Discovery;
using Energistics.Protocol.Store;
using Energistics.Security;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Energistics
{
    [TestClass]
    public class EtpClientTests
    {
        private const string Uri = "wss://witsml.pds.nl/api/etp";
        private const string AppName = "EtpClientTests";
        private const string AppVersion = "1.0";

        [TestMethod]
        public async Task EtpClient_Opens_WebSocket_Connection()
        {
            // Create a Basic authorization header dictionary
            var auth = Authorization.Basic("witsml.user", "P@$$^0rd!");

            // Initialize an EtpClient with a valid Uri, app name and version, and auth header
            using (var client = new EtpClient(Uri, AppName, AppVersion, auth))
            {
                // Register protocol handlers to be used in later tests
                client.Register<IChannelStreamingConsumer, ChannelStreamingConsumerHandler>();
                client.Register<IDiscoveryCustomer, DiscoveryCustomerHandler>();
                client.Register<IStoreCustomer, StoreCustomerHandler>();

                // Open the connection (uses an async extension method)
                await client.OpenAsync();

                // Assert the current state of the connection
                Assert.IsTrue(client.IsOpen);

                // Explicit Close not needed as the WebSocket connection will be closed
                // automatically after leaving the scope of the using statement
                //client.Close("reason");
            }
        }
    }
}
```

The code comments explain each step in the process and mention the use of an OpenAsync extension method, which is explained in the next section.

4.2   Test Extensions

The following class defines the extension methods that will be utilized with the examples to execute and inspect the messages received asynchronously.

```csharp
using System;
using System.Threading;
using System.Threading.Tasks;

namespace Energistics
{
    /// <summary>
    /// Provides static helper methods that can be used to process ETP messages asynchronously.
    /// </summary>
    public static class TestExtensions
    {
        /// <summary>
        /// Opens a WebSocket connection and waits for the SocketOpened event to be called.
        /// </summary>
        /// <param name="client">The client.</param>
        /// <returns>An awaitable task.</returns>
        public static async Task<bool> OpenAsync(this EtpClient client)
        {
            var task = new Task<bool>(() => client.IsOpen);

            client.SocketOpened += (s, e) => task.Start();
            client.Open();

            return await task.WaitAsync();
        }

        /// <summary>
        /// Executes a task asynchronously and waits the specified timeout period for it to complete.
        /// </summary>
        /// <typeparam name="TResult">The type of the result.</typeparam>
        /// <param name="task">The task to execute.</param>
        /// <param name="milliseconds">The timeout, in milliseconds.</param>
        /// <returns>An awaitable task.</returns>
        /// <exception cref="System.TimeoutException">The operation has timed out.</exception>
        public static async Task<TResult> WaitAsync<TResult>(this Task<TResult> task, int milliseconds = 5000)
        {
            return await task.WaitAsync(TimeSpan.FromMilliseconds(milliseconds));
        }

        /// <summary>
        /// Executes a task asynchronously and waits the specified timeout period for it to complete.
        /// </summary>
        /// <typeparam name="TResult">The type of the result.</typeparam>
        /// <param name="task">The task to execute.</param>
        /// <param name="timeout">The timeout.</param>
        /// <returns>An awaitable task.</returns>
        /// <exception cref="System.TimeoutException">The operation has timed out.</exception>
        public static async Task<TResult> WaitAsync<TResult>(this Task<TResult> task, TimeSpan timeout)
        {
            var tokenSource = new CancellationTokenSource();
            var completedTask = await Task.WhenAny(task, Task.Delay(timeout, tokenSource.Token));

            if (completedTask == task)
            {
                tokenSource.Cancel();
                return await task;
            }

            throw new TimeoutException("The operation has timed out.");
        }
    }
}
```

4.3 Configuration Settings

To help make testing easier, the EtpClient connection parameters can be moved to the application settings area of the project properties.

	Name	Type		Scope		Value
▶	AuthTokenUrl	string	∨	Application	∨	https://witsml.pds.nl/auth
	ServerUrl	string	∨	Application	∨	wss://witsml.pds.nl/api/etp
	Username	string	∨	Application	∨	witsml.user
	Password	string	∨	Application	∨	P@$$^0rd!
＊			∨		∨	

4.4 Integration Test Setup

The configuration settings, along with additional helper methods can be brought together in a base class which can be used as the foundation for all future testing of client-server interaction.

```csharp
using System;
using System.Threading.Tasks;
using Avro.Specific;
using Energistics.Common;
using Energistics.IntegrationTest;
using Energistics.Security;

namespace Energistics
{
    /// <summary>
    /// Common base class for all ETP DevKit integration tests.
    /// </summary>
    public abstract class IntegrationTestBase
    {
        protected static readonly string AuthTokenUrl = Settings.Default.AuthTokenUrl;
        protected static readonly string ServerUrl = Settings.Default.ServerUrl;
        protected static readonly string Username = Settings.Default.Username;
        protected static readonly string Password = Settings.Default.Password;

        /// <summary>
        /// Creates an <see cref="EtpClient"/> instance configured with the
        /// current connection and authorization parameters.
        /// </summary>
        /// <returns></returns>
        protected EtpClient CreateClient()
        {
            var version = GetType().Assembly.GetName().Version.ToString();
            var headers = Authorization.Basic(Username, Password);

            return new EtpClient(ServerUrl, GetType().AssemblyQualifiedName, version, headers);
        }

        /// <summary>
        /// Handles an event asynchronously and waits for it to complete.
        /// </summary>
        /// <typeparam name="T">The type of ETP message.</typeparam>
        /// <param name="action">The action to execute.</param>
        /// <returns>An awaitable task.</returns>
        protected async Task<ProtocolEventArgs<T>> HandleAsync<T>(Action<ProtocolEventHandler<T>> action)
            where T : ISpecificRecord
        {
            ProtocolEventArgs<T> args = null;
            var task = new Task<ProtocolEventArgs<T>>(() => args);

            action((s, e) =>
            {
                args = e;

                if (task.Status == TaskStatus.Created)
                    task.Start();
            });
```

```
            return await task.WaitAsync();
        }

        /// <summary>
        /// Handles an event asynchronously and waits for it to complete.
        /// </summary>
        /// <typeparam name="T">The type of ETP message.</typeparam>
        /// <typeparam name="TContext">The type of the context.</typeparam>
        /// <param name="action">The action to execute.</param>
        /// <returns>An awaitable task.</returns>
        protected async Task<ProtocolEventArgs<T, TContext>> HandleAsync<T, TContext>(
            Action<ProtocolEventHandler<T, TContext>> action)
            where T : ISpecificRecord
        {
            ProtocolEventArgs<T, TContext> args = null;
            var task = new Task<ProtocolEventArgs<T, TContext>>(() => args);

            action((s, e) =>
            {
                args = e;

                if (task.Status == TaskStatus.Created)
                    task.Start();
            });

            return await task.WaitAsync();
        }
    }
}
```

4.5   Connect to a Simple Producer

For this test, we will utilize the IntegrationTestBase and the extension methods mentioned previously to cut down on the amount of code needed to initialize a test.

```
using System.Linq;
using System.Threading.Tasks;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Energistics.Protocol.ChannelStreaming
{
    [TestClass]
    public class ChannelStreamingProtocolTests : IntegrationTestBase
    {
        private EtpClient _client;

        [TestInitialize]
        public void TestSetUp()
        {
            _client = CreateClient();
        }

        [TestCleanup]
        public void TestTearDown()
        {
            _client.Dispose();
        }

        [TestMethod]
        public async Task IChannelStreamingConsumer_Start_Connected_To_Simple_Producer()
        {
            // Register protocol handler
            _client.Register<IChannelStreamingConsumer, ChannelStreamingConsumerHandler>();
            var handler = _client.Handler<IChannelStreamingConsumer>();

            // Register event handlers
            var onChannelMetadata = HandleAsync<ChannelMetadata>(x => handler.OnChannelMetadata += x);
            var onChannelData = HandleAsync<ChannelData>(x => handler.OnChannelData += x);

            // Wait for Open connection
            var isOpen = await _client.OpenAsync();
            Assert.IsTrue(isOpen);
```

```
            // Send Start message
            handler.Start();

            // Wait for ChannelMetadata message
            var argsMetadata = await onChannelMetadata.WaitAsync();

            Assert.IsNotNull(argsMetadata);
            Assert.IsNotNull(argsMetadata.Message.Channels);
            Assert.IsTrue(argsMetadata.Message.Channels.Any());

            // Wait for ChannelData message
            var argsData = await onChannelData.WaitAsync();

            Assert.IsNotNull(argsData);
            Assert.IsNotNull(argsData.Message.Data);
            Assert.IsTrue(argsData.Message.Data.Any());
        }
    }
}
```

This integration test registers the default ChannelStreamingConsumerHandler as the protocol handler for the consumer role of the Channel Streaming protocol. It also utilizes the HandleAsync<T> method defined in the IntegrationTestBase class to create awaitable Task instances that complete when the corresponding events are raised.

After the EtpClient instance opens the Web Socket connection, the handler sends the Start message and then waits for the ChannelMetadata and ChannelData messages from the producer.

4.6   Using the Discovery Protocol

With the Discovery protocol handler, we can write a test that shows how to process the messages received from a Store to simulate the two-way interaction found in most client applications.

```
using System.Threading.Tasks;
using Energistics.Datatypes;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Energistics.Protocol.Discovery
{
    [TestClass]
    public class DiscoveryProtocolTests : IntegrationTestBase
    {
        private EtpClient _client;

        [TestInitialize]
        public void TestSetUp()
        {
            _client = CreateClient();
        }

        [TestCleanup]
        public void TestTearDown()
        {
            _client.Dispose();
        }

        [TestMethod]
        public async Task IDiscoveryCustomer_GetResource_Request_Default_Uri()
        {
            // Register protocol handler
            _client.Register<IDiscoveryCustomer, DiscoveryCustomerHandler>();
            var handler = _client.Handler<IDiscoveryCustomer>();

            // Wait for Open connection
            var isOpen = await _client.OpenAsync();
            Assert.IsTrue(isOpen);

            // Register event handler for root URI
            var onGetRootResourcesResponse = HandleAsync<GetResourcesResponse, string>(
                x => handler.OnGetResourcesResponse += x);
```

[14]

```
            // Send GetResources message for root URI
            handler.GetResources(EtpUri.RootUri);

            // Wait for GetResourcesResponse for top level resources
            var argsRoot = await onGetRootResourcesResponse.WaitAsync();

            Assert.IsNotNull(argsRoot);
            Assert.IsNotNull(argsRoot.Message.Resource);
            Assert.IsNotNull(argsRoot.Message.Resource.Uri);

            // Register event handler for child resources
            var onGetChildResourcesResponse = HandleAsync<GetResourcesResponse, string>(
                x => handler.OnGetResourcesResponse += x);

            // Send GetResources message for child resources
            var resource = argsRoot.Message.Resource;
            handler.GetResources(resource.Uri);

            // Wait for GetResourcesResponse for child resources
            var argsChild = await onGetChildResourcesResponse.WaitAsync();

            Assert.IsNotNull(argsChild);

            if (argsChild.Header.MessageFlags == (int) MessageFlags.NoData)
            {
                Assert.IsNull(argsChild.Message.Resource);
            }
            else
            {
                Assert.IsNotNull(argsChild.Message.Resource);
                Assert.AreNotEqual(resource.Uri, argsChild.Message.Resource.Uri);
            }
        }
    }
}
```

In this example, multiple event handlers are registered in order to asynchronously process the response from the same message type multiple times.

5 Sample Client Application

The previous examples all showed how to test ETP connectivity using the ETP DevKit and several integration test helper methods. Now, we will implement a simple console application that will demonstrate a more real-world example of an ETP client. Note, the full source code for the sample client and server application can be found in the TestApp project found in the solution downloaded from the Git repository.

Create a new console application project in the ETP solution and set the target framework to ".NET Framework 4.5.2". Add a project reference for the DevKit project and add a NuGet package reference to ETP (version 1.5.0) and log4net.

Create a log4net.config file in the TestApp root folder with the following contents.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<log4net>
  <appender name="RollingFileAppender" type="log4net.Appender.RollingFileAppender">
    <file value="TestApp.log" />
    <appendToFile value="true" />
    <rollingStyle value="Size" />
    <maxSizeRollBackups value="10" />
    <maximumFileSize value="4096KB" />
    <staticLogFileName value="true" />
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%date [%thread] %-5level %logger - %message%newline" />
    </layout>
  </appender>
  <appender name="ConsoleAppender" type="log4net.Appender.ConsoleAppender">
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%date [%thread] %-5level %logger %M - %message%newline" />
    </layout>
  </appender>
  <root>
    <level value="ALL" />
    <appender-ref ref="RollingFileAppender" />
    <appender-ref ref="ConsoleAppender" />
  </root>
</log4net>
```

Next, replace the contents of the Program.cs file with the following code:

```csharp
using System;
using Energistics.Common;
using Energistics.Protocol.ChannelStreaming;
using Energistics.Protocol.Discovery;
using log4net.Config;

namespace Energistics
{
    public class Program
    {
        private static readonly string AppVersion = typeof(Program).Assembly.GetName().Version.ToString();
        private const string ClientAppName = "etp-client";

        public static void Main(string[] args)
        {
            XmlConfigurator.ConfigureAndWatch(new FileInfo("log4net.config"));
            StartClient();
        }

        private static void StartClient()
        {
            Console.WriteLine("Enter a valid Web Socket URI:");
            var webSocketUri = Console.ReadLine();
            Console.WriteLine();

            Console.WriteLine("Select from the following options:");
            Console.WriteLine(" O - open");
            Console.WriteLine(" C - close");
            Console.WriteLine(" Z - clear");
            Console.WriteLine(" X - exit");
```

[16]

```csharp
            Console.WriteLine(" S - ChannelStreaming - Start");
            Console.WriteLine(" D - Discovery - GetResources");
            Console.WriteLine();

            using (var client = new EtpClient(webSocketUri, ClientAppName, AppVersion))
            {
                // Register protocol handlers
                client.Register<IChannelStreamingConsumer, ChannelStreamingConsumerHandler>();
                client.Register<IDiscoveryCustomer, DiscoveryCustomerHandler>();
                client.Handler<IDiscoveryCustomer>().OnGetResourcesResponse += OnGetResourcesResponse;

                while (true)
                {
                    var info = Console.ReadKey();

                    Console.WriteLine(" - processing...");
                    Console.WriteLine();

                    if (IsKey(info, "O"))
                    {
                        client.Open();
                    }
                    else if (IsKey(info, "C"))
                    {
                        client.Close("EtpClient closed.");
                    }
                    else if (IsKey(info, "S"))
                    {
                        Console.WriteLine("Starting ChannelStreaming session...");
                        client.Handler<IChannelStreamingConsumer>()
                            .Start(maxMessageRate: 2000);
                    }
                    else if (IsKey(info, "D"))
                    {
                        Console.WriteLine("Enter resource URI:");
                        var uri = Console.ReadLine();
                        Console.WriteLine();

                        client.Handler<IDiscoveryCustomer>()
                            .GetResources(uri);
                    }
                    else if (IsKey(info, "Z"))
                    {
                        Console.Clear();
                    }
                    else if (IsKey(info, "X"))
                    {
                        break;
                    }
                }
            }
        }

        private static void OnGetResourcesResponse(object sender, ProtocolEventArgs<GetResourcesResponse> e)
        {
            Console.WriteLine(((EtpBase)sender).Serialize(e.Message.Resource, true));
        }

        private static bool IsKey(ConsoleKeyInfo info, string key)
        {
            return string.Equals(info.KeyChar.ToString(), key, StringComparison.InvariantCultureIgnoreCase);
        }
    }
}
```

After instantiating the EtpClient instance in the StartClient method, notice that we have registered the default protocol handlers for the ChannelStreaming consumer and Discovery customer roles, but we only registered an event handler for the OnGetResourcesResponse event of the IDiscoveryCustomer interface.

To do anything meaningful with the messages received from a Channel Streaming producer, we will need to implement a custom protocol handler for IChannelStreamingConsumer. To demonstrate this, create a new folder in the console

[17]

application project named Providers and create a new class called MockChannelStreamingConsumer and paste the following code as the content of the file:

```csharp
using System;
using System.Linq;
using Energistics.Common;
using Energistics.Datatypes;
using Energistics.Protocol.ChannelStreaming;

namespace Energistics.Providers
{
    /// <summary>
    /// Custom implementation of <see cref="IChannelStreamingConsumer"/> for connecting to a Simple Producer
    /// </summary>
    /// <seealso cref="Energistics.Protocol.ChannelStreaming.ChannelStreamingConsumerHandler" />
    public class MockChannelStreamingConsumer : ChannelStreamingConsumerHandler
    {
        /// <summary>
        /// Handles the ChannelMetadata message from a producer.
        /// </summary>
        /// <param name="header">The message header.</param>
        /// <param name="channelMetadata">The ChannelMetadata message.</param>
        protected override void HandleChannelMetadata(MessageHeader header, ChannelMetadata channelMetadata)
        {
            Console.WriteLine(string.Join(Environment.NewLine, channelMetadata.Channels.Select(this.Serialize)));
        }

        /// <summary>
        /// Handles the ChannelData message from a producer.
        /// </summary>
        /// <param name="header">The message header.</param>
        /// <param name="channelData">The ChannelData message.</param>
        protected override void HandleChannelData(MessageHeader header, ChannelData channelData)
        {
            Console.WriteLine(string.Join(Environment.NewLine, channelData.Data.Select(this.Serialize)));
        }
    }
}
```

For demo purposes, this implementation will simply output the messages it receives to the console. To hook it up to the EtpClient instance, update the protocol registration as follows. Note: be sure to include the appropriate using statement to make the new class available to the program.

```csharp
using Energistics.Providers;

// Register protocol handlers
client.Register<IChannelStreamingConsumer, ChannelStreamingConsumerHandler>();
client.Register<IChannelStreamingConsumer, MockChannelStreamingConsumer>();
```
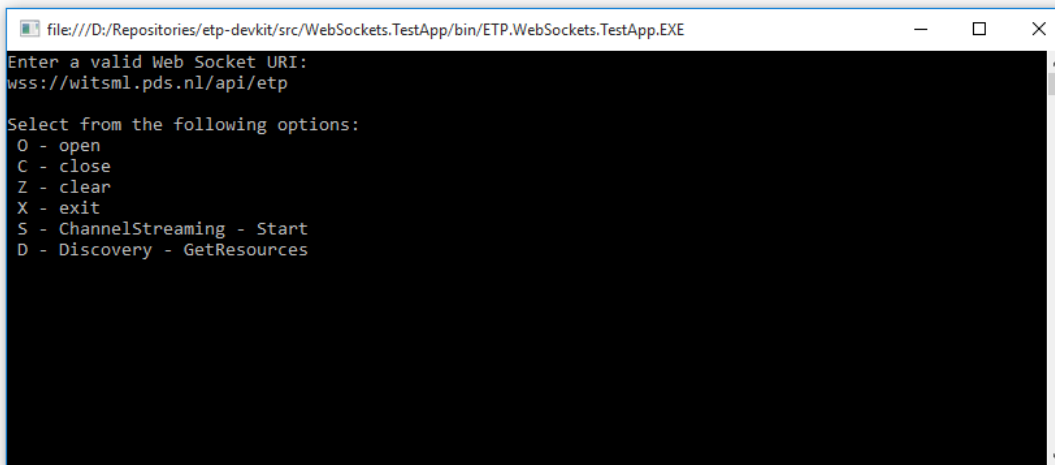
Now, we can compile and run the application to connect to a Simple Streamer by following the prompts on the screen:
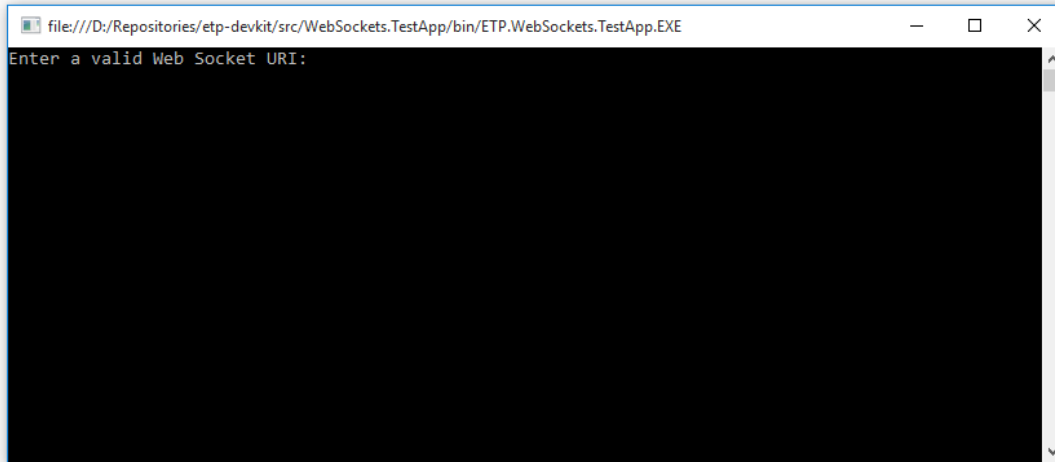
After entering a valid Web Socket URI, you will be prompted to select from a list of available options. You will need to open the connection before attempting to use any of the registered protocol handlers.

Once the connection is established, if you are connecting to a Simple Streamer, pressing S + Enter will begin the Channel Streaming session and you should start seeing ChannelMetadata and ChannelData messages printed to the console as they are received.

file:///D:/Repositories/etp-devkit/src/WebSockets.TestApp/bin/ETP.WebSockets.TestApp.EXE

2016-05-09 19:26:11,176 [19] DEBUG Energistics.EtpClient Decode - [e6473c97-ad35-4e8d-ab7b-1bfea92746a3] Message received: {"protocol":1,
"messageType":3,"correlationId":0,"messageId":11,"messageFlags":0}
{"indexes":[1462839970087000],"channelId":0,"value":{"item":1.0},"valueAttributes":[]}
{"indexes":[],"channelId":1,"value":{"item":1.0},"valueAttributes":[]}
{"indexes":[],"channelId":2,"value":{"item":0.0},"valueAttributes":[]}
{"indexes":[],"channelId":3,"value":{"item":0.0},"valueAttributes":[]}
{"indexes":[],"channelId":4,"value":{"item":0.0},"valueAttributes":[]}
{"indexes":[],"channelId":5,"value":{"item":0.0},"valueAttributes":[]}
{"indexes":[],"channelId":6,"value":{"item":92.0},"valueAttributes":[]}
{"indexes":[],"channelId":7,"value":{"item":92.0},"valueAttributes":[]}
{"indexes":[],"channelId":8,"value":{"item":0.0},"valueAttributes":[]}
{"indexes":[],"channelId":9,"value":{"item":0.0},"valueAttributes":[]}
{"indexes":[],"channelId":10,"value":{"item":0.0},"valueAttributes":[]}
{"indexes":[],"channelId":11,"value":{"item":0.0},"valueAttributes":[]}
{"indexes":[],"channelId":12,"value":{"item":6.0},"valueAttributes":[]}
{"indexes":[],"channelId":13,"value":{"item":6.0},"valueAttributes":[]}
{"indexes":[],"channelId":14,"value":{"item":0.0},"valueAttributes":[]}
{"indexes":[],"channelId":15,"value":{"item":0.0},"valueAttributes":[]}
{"indexes":[],"channelId":16,"value":{"item":99.0},"valueAttributes":[]}
{"indexes":[],"channelId":17,"value":{"item":99.0},"valueAttributes":[]}
{"indexes":[],"channelId":18,"value":{"item":0.0},"valueAttributes":[]}
{"indexes":[],"channelId":19,"value":{"item":0.0},"valueAttributes":[]}
{"indexes":[],"channelId":20,"value":{"item":99.0},"valueAttributes":[]}
{"indexes":[],"channelId":21,"value":{"item":99.0},"valueAttributes":[]}
{"indexes":[],"channelId":22,"value":{"item":0.0},"valueAttributes":[]}
{"indexes":[],"channelId":23,"value":{"item":0.0},"valueAttributes":[]}
{"indexes":[],"channelId":24,"value":{"item":0.0},"valueAttributes":[]}
{"indexes":[],"channelId":25,"value":{"item":0.0},"valueAttributes":[]}
{"indexes":[],"channelId":26,"value":{"item":48.0},"valueAttributes":[]}
{"indexes":[],"channelId":27,"value":{"item":48.0},"valueAttributes":[]}
{"indexes":[],"channelId":28,"value":{"item":0.0},"valueAttributes":[]}
{"indexes":[],"channelId":29,"value":{"item":0.0},"valueAttributes":[]}
c - processing...

2016-05-09 19:26:11,640 [8] DEBUG Energistics.EtpClient Close - Closing web socket connection: EtpClient closed.
2016-05-09 19:26:11,656 [19] DEBUG Energistics.EtpClient OnWebSocketClosed - [e6473c97-ad35-4e8d-ab7b-1bfea92746a3] Socket closed.

Press C to stop streaming and/or close the connection. X will exit the application.

6 Sample Server Application

For our Web Socket server example, we will build upon the sample console application we started in the previous section. Note, the full source code for the sample client and server application can be found in the TestApp project found in the solution downloaded from the Git repository.

To get started, add a NuGet reference to SuperWebSocket (version 0.9.0.2).

Then open the Program.cs file to add the following constants and new methods, and also update the Main method to call Start instead of StartClient.

```csharp
private const string ServerAppName = "etp-server";

private const string WebSocketUri = "ws://localhost:9000";
private const int WebSocketPort = 9000;

public static void Main(string[] args)
{
    XmlConfigurator.ConfigureAndWatch(new FileInfo("log4net.config"));
    Start();
}

private static void Start()
{
    Console.Write("Press 'S' to start a web socket server,");
    Console.WriteLine(" or press 'C' to start a client instance...");

    var key = Console.ReadKey();

    Console.WriteLine(" - processing...");
    Console.WriteLine();

    if (IsKey(key, "S"))
    {
        StartServer();
    }
    else if (IsKey(key, "C"))
    {
        StartClient();
    }
    else
    {
        Start();
    }
}

private static void StartServer()
{
    Console.WriteLine("Select from the following options:");
    Console.WriteLine(" S - start / stop");
    Console.WriteLine(" Z - clear");
    Console.WriteLine(" X - exit");
    Console.WriteLine();

    using (var server = new EtpSocketServer(WebSocketPort, ServerAppName, AppVersion))
    {
        // Register protocol handlers
        server.Register<IDiscoveryStore, MockResourceProvider>();

        while (true)
        {
            var info = Console.ReadKey();

            Console.WriteLine(" - processing...");
            Console.WriteLine();

            if (IsKey(info, "S"))
            {
                if (server.IsRunning)
                    server.Stop();
                else
                    server.Start();
```

```
            }
            else if (IsKey(info, "Z"))
            {
                Console.Clear();
            }
            else if (IsKey(info, "X"))
            {
                break;
            }
        }
    }
}
```

We will implement the MockResourceProvider class to enable the EtpSocketServer to support the Discovery protocol's store role in order to respond to a customer's GetResources message. Create a new class in the Providers folder named MockResourceProvider and copy-and-paste the following code, or copy the file from the TestApp project.

```csharp
using System;
using System.Collections.Generic;
using Energistics.Common;
using Energistics.Datatypes;
using Energistics.Datatypes.Object;
using Energistics.Protocol.Discovery;

namespace Energistics.Providers
{
    public class MockResourceProvider : DiscoveryStoreHandler
    {
        private const string Witsml141 = "application/x-witsml+xml;version=1.4.1.1;";
        private const string BaseUri = "eml://witsml14";

        protected override void HandleGetResources(ProtocolEventArgs<GetResources, IList<Resource>> args)
        {
            if (EtpUri.IsRoot(args.Message.Uri))
            {
                args.Context.Add(New(
                    x => BaseUri,
                    contentType: Witsml141,
                    resourceType: ResourceTypes.UriProtocol,
                    name: "WITSML Store (1.4.1.1)"));
            }
            else if (EtpUri.IsRoot(args.Message.Uri))
            {
                args.Context.Add(New(
                    uuid => String.Format("{0}/well({1})", args.Message.Uri, uuid),
                    contentType: Witsml141 + "type=obj_well",
                    resourceType: ResourceTypes.DataObject,
                    name: "Well 01"));

                args.Context.Add(New(
                    uuid => String.Format("{0}/well({1})", args.Message.Uri, uuid),
                    contentType: Witsml141 + "type=obj_well",
                    resourceType: ResourceTypes.DataObject,
                    name: "Well 02"));
            }
            else if (args.Message.Uri.Contains("/well(") && !args.Message.Uri.Contains("/wellbore("))
            {
                args.Context.Add(New(
                    uuid => String.Format("{0}/wellbore({1})", args.Message.Uri, uuid),
                    contentType: Witsml141 + "type=obj_wellbore",
                    resourceType: ResourceTypes.DataObject,
                    name: "Wellbore 01-01"));

                args.Context.Add(New(
                    uuid => String.Format("{0}/wellbore({1})", args.Message.Uri, uuid),
                    contentType: Witsml141 + "type=obj_wellbore",
                    resourceType: ResourceTypes.DataObject,
                    name: "Wellbore 01-02"));
            }
            else if (args.Message.Uri.Contains("/wellbore("))
            {
                args.Context.Add(New(
```

```
                uuid => String.Format("{0}/log({1})", args.Message.Uri, uuid),
                contentType: Witsml141 + "type=obj_log",
                resourceType: ResourceTypes.DataObject,
                name: "Depth Log 01",
                count: 0));

            args.Context.Add(New(
                uuid => String.Format("{0}/log({1})", args.Message.Uri, uuid),
                contentType: Witsml141 + "type=obj_log",
                resourceType: ResourceTypes.DataObject,
                name: "Time Log 01",
                count: 0));
        }
    }

    private Resource New(Func<string, string> formatUri, ResourceTypes resourceType, string contentType,
                        string name, int count = 1)
    {
        var uuid = Guid.NewGuid().ToString();

        return new Resource()
        {
            Uuid = uuid,
            Uri = formatUri(uuid),
            Name = name,
            HasChildren = count,
            ContentType = contentType,
            ResourceType = resourceType.ToString(),
            CustomData = new Dictionary<string, string>()
        };
    }
    }
}
```

Compile and run the console application to start an instance of the ETP Web Socket Server. You will be prompted to choose S for server (or C for client), followed by additional options to start/stop the server, clear the console or exit the application.

Any messages sent or received will be logged to the console window.

A second instance of the console application can be run as an ETP Client to connect to the sample server. The URL to connect to the server will be "ws://localhost:9000".

The following screen shots illustrate the request messages sent from the client and the corresponding response messages sent from the server.


```
C:\Windows\system32\cmd.exe                                                           —    □    ×
Press 'S' to start a web socket server, or press 'C' to start a client instance...
c - processing...

Enter a valid Web Socket URI [ws://localhost:9000]:
ws://localhost:9000

Select from the following options:
 O - open
 C - close
 Z - clear
 X - exit
 S - ChannelStreaming - Start
 D - Discovery - GetResources

o - processing...

2016-05-10 10:57:57,868 [1] DEBUG Energistics.EtpClient Open - Opening web socket connection...
2016-05-10 10:57:58,962 [3] DEBUG Energistics.EtpClient OnWebSocketOpened - [] Socket opened.
2016-05-10 10:57:59,150 [3] DEBUG Energistics.EtpClient Sent - [] Message sent: {"protocol":0,"messageType":1,"correlationId":0,"messageId":1,"messageFlags":0}
2016-05-10 10:57:59,332 [3] DEBUG Energistics.EtpClient Decode - [] Message received: {"protocol":0,"messageType":2,"correlationId":1,"messageId":1,"messageFlags":0}
d - processing...

Enter resource URI:
/

2016-05-10 10:58:22,341 [1] DEBUG Energistics.EtpClient Sent - [c76cfcbe-55c5-4292-acee-049122281231] Message sent: {"protocol":3,"messageType":1,"correlationId":0,"messageId":2,"messageFlags":0}
2016-05-10 10:58:22,349 [5] DEBUG Energistics.EtpClient Decode - [c76cfcbe-55c5-4292-acee-049122281231] Message received: {"protocol":3,"messageType":2,"correlationId":2,"messageId":2,"messageFlags":2}
{
  "uri": "eml://witsml14",
  "contentType": "application/x-witsml+xml;version=1.4.1.1;",
  "name": "WITSML Store (1.4.1.1)",
  "channelSubscribable": false,
  "customData": {},
  "resourceType": "UriProtocol",
  "hasChildren": 1,
  "uuid": "07c40035-1e7e-4caa-9859-b02ef3dc789e",
  "lastChanged": 0,
  "objectNotifiable": false
}
```



```
C:\Windows\system32\cmd.exe                                                           —    □    ×
Press 'S' to start a web socket server, or press 'C' to start a client instance...
s - processing...

Select from the following options:
 S - start / stop
 Z - clear
 X - exit

s - processing...

2016-05-10 10:57:39,772 [1] DEBUG WebSocketServer-26753075 DebugFormat - Listener (0.0.0.0:9000) was started
2016-05-10 10:57:39,798 [1] INFO  WebSocketServer-26753075 Info - The server instance WebSocketServer-26753075 has been started!
2016-05-10 10:57:58,960 [5] DEBUG Energistics.EtpSocketServer OnNewSessionConnected - [c76cfcbe-55c5-4292-acee-049122281231] Socket session connected.
2016-05-10 10:57:59,229 [8] DEBUG Energistics.EtpServer Decode - [c76cfcbe-55c5-4292-acee-049122281231] Message received: {"protocol":0,"messageType":1,"correlationId":0,"messageId":1,"messageFlags":0}
2016-05-10 10:57:59,319 [8] DEBUG Energistics.EtpServer Sent - [c76cfcbe-55c5-4292-acee-049122281231] Message sent: {"protocol":0,"messageType":2,"correlationId":1,"messageId":1,"messageFlags":0}
2016-05-10 10:58:22,341 [5] DEBUG Energistics.EtpServer Decode - [c76cfcbe-55c5-4292-acee-049122281231] Message received: {"protocol":3,"messageType":1,"correlationId":0,"messageId":2,"messageFlags":0}
2016-05-10 10:58:22,348 [5] DEBUG Energistics.EtpServer Sent - [c76cfcbe-55c5-4292-acee-049122281231] Message sent: {"protocol":3,"messageType":2,"correlationId":2,"messageId":2,"messageFlags":2}
 - processing...
```

7 References

The following documents and source code repositories were referenced during development of the ETP DevKit library.

- ETP Specification.docx
- Energistics Identifier Specification.docx
- ETP_Implementation_Guide.docx
- ETP Node Server - https://bitbucket.org/energistics/etp-server-js
- ETP Prototype for C# - https://bitbucket.org/energistics/etp-samples