ETP DevKit – Getting Started

1 Introduction

This guide introduces the ETP DevKit library and gives examples for both client and server scenarios.

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 4.6

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.

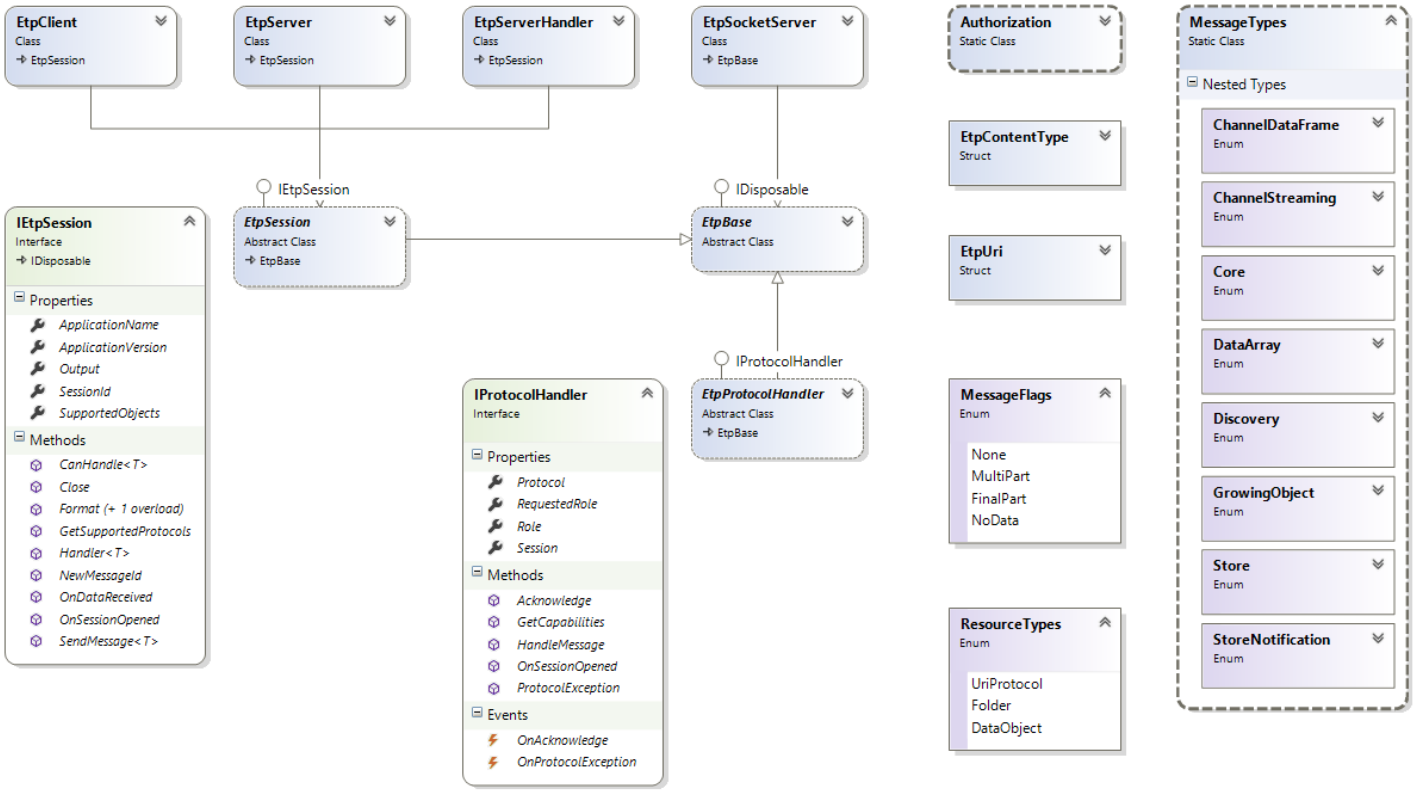
Each interface implementation has been developed as a protocol handler that can be used out of the box or extended to provide additional functionality. Customized processing of messages can be achieved by either registering handlers for the various interface events or by deriving from the library's protocol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

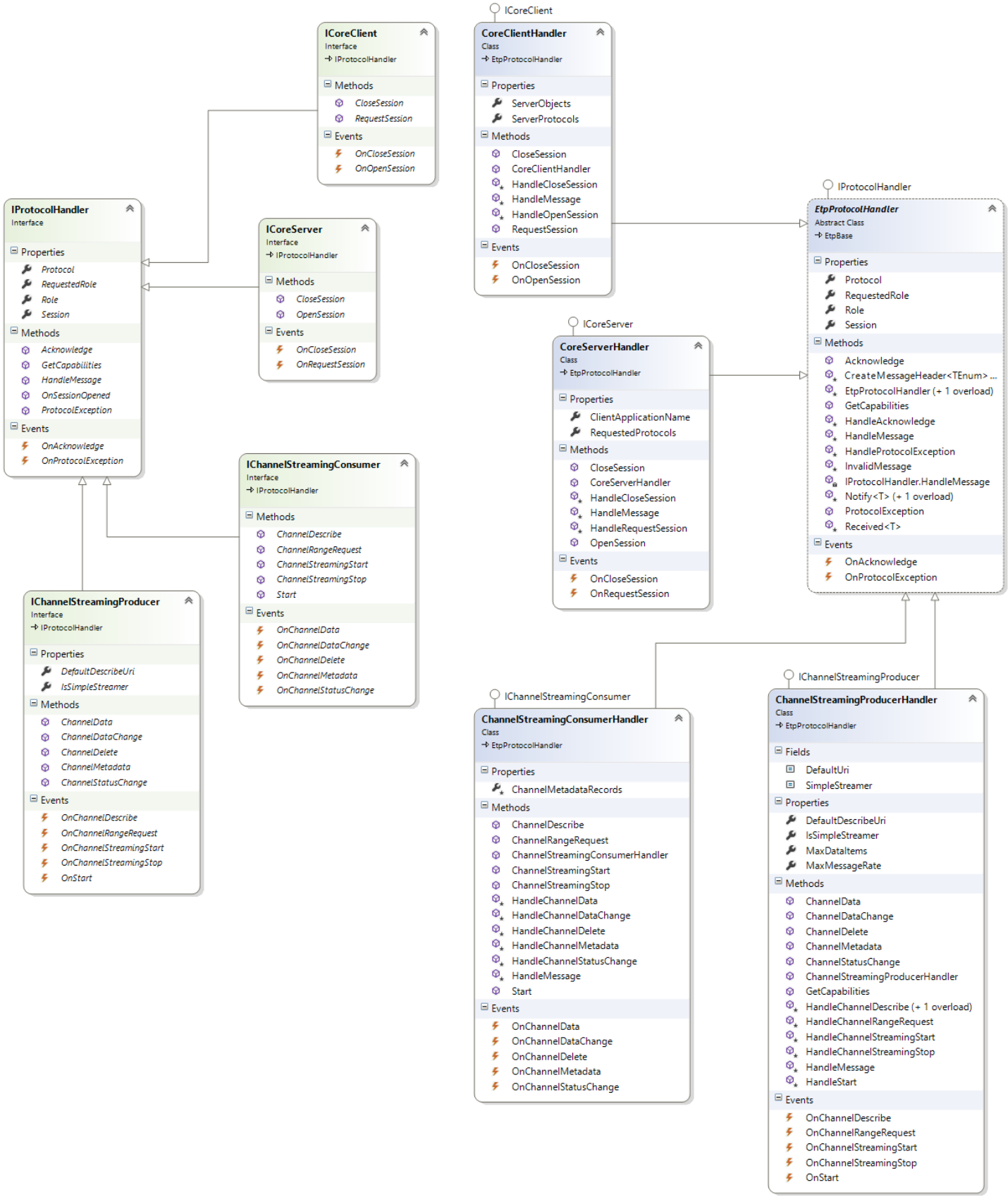
2.1 Class Diagrams

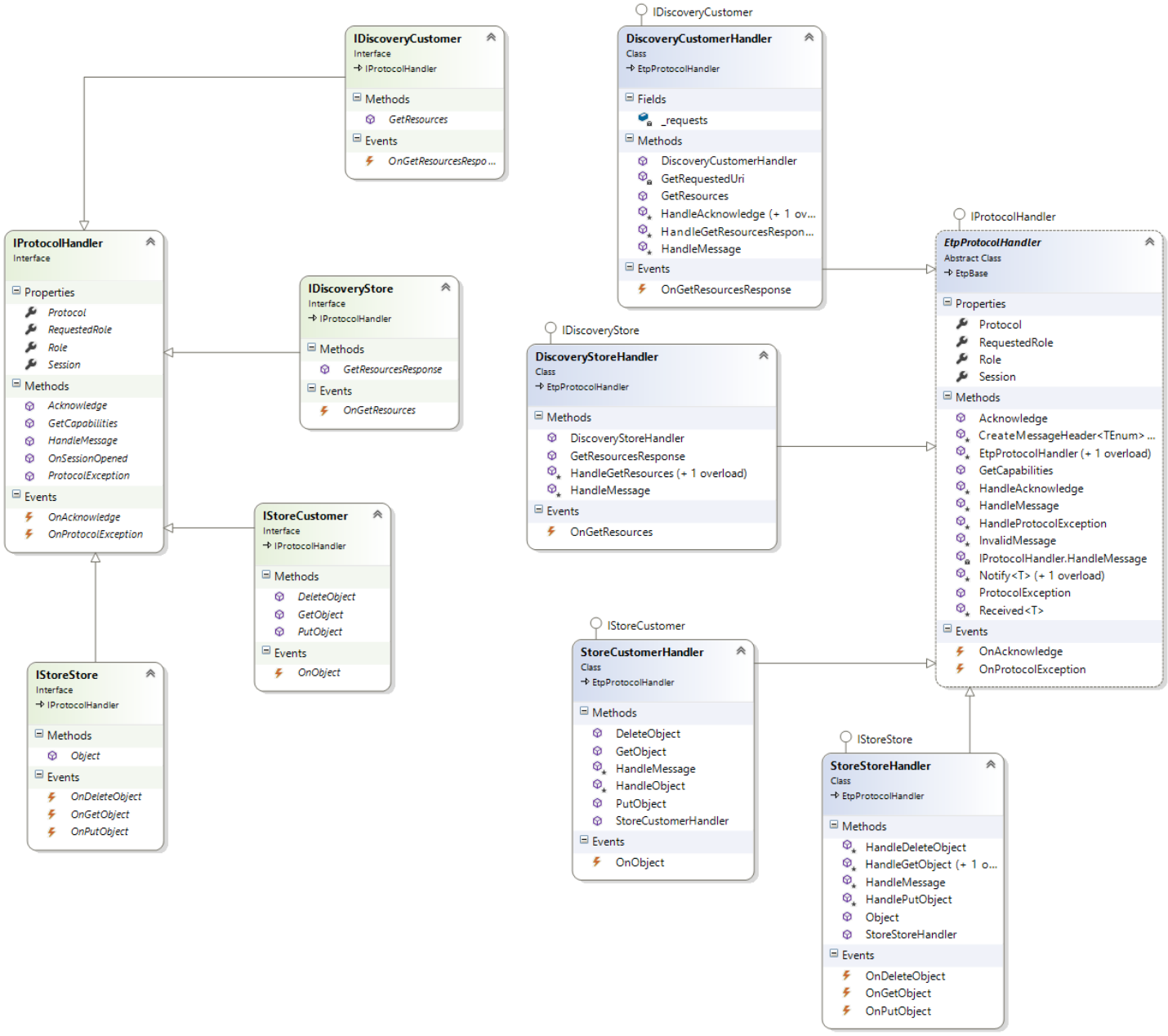
The following class diagrams provide an overview of the types found in the ETP DevKit library.

2.1.1 Base Types



2.1.2 Protocol Handlers

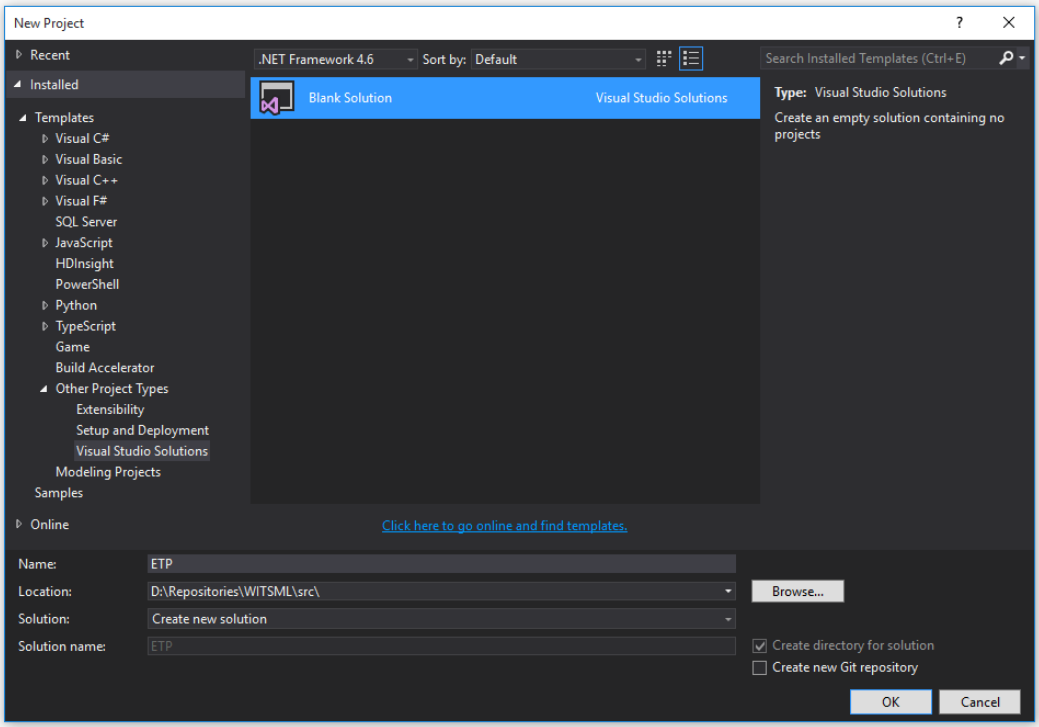



IDiscoveryCustomer
Interface
IProtocolHandler
Methods
GetResources
Events
OnGetResourcesRespo ...
IDiscoveryCustomer
DiscoveryCustomerHandler
Class
EtpProtocolHandler
Fields
_requests
Methods
DiscoveryCustomerHandler
GetRequestedUri
GetResources
HandleAcknowledge (+ 1 ov...
HandleGetResourcesRespon...
HandleMessage
Events
OnGetResourcesResponse
IProtocolHandler
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
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
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

3 Solution and Project Setup

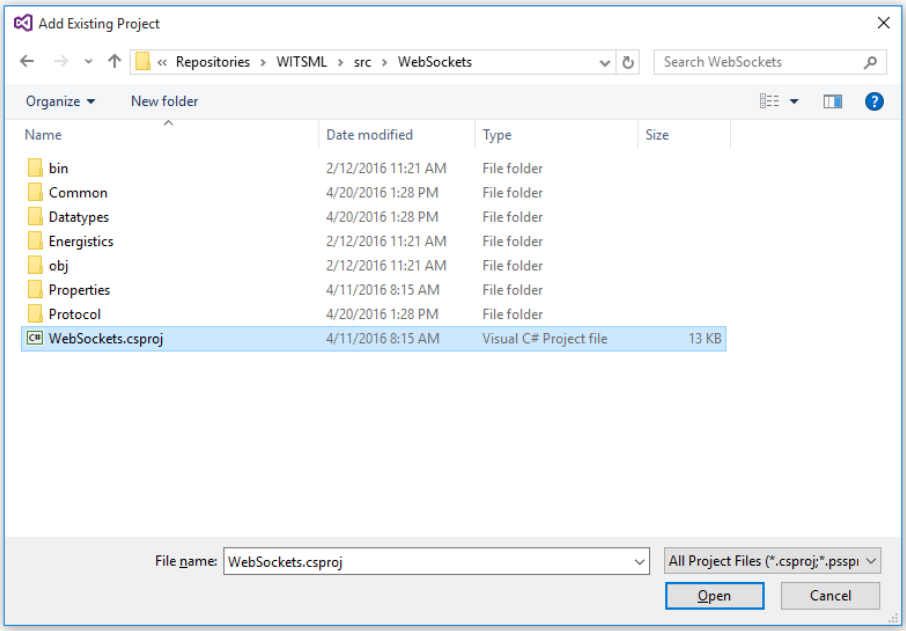
3.1 New Solution

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



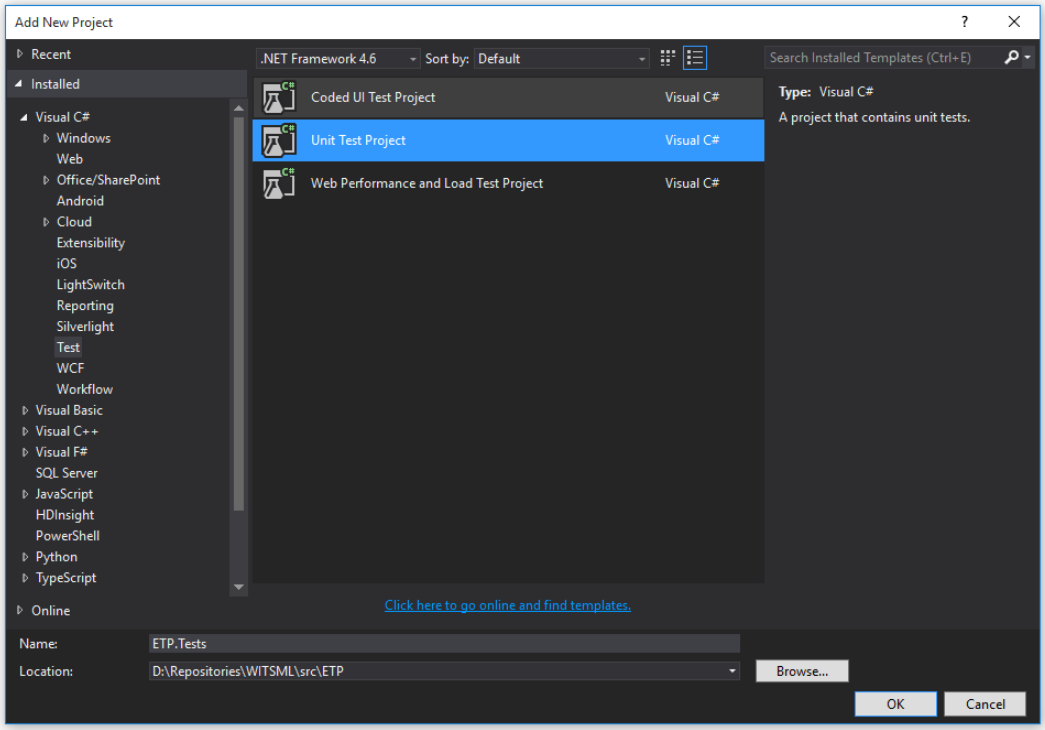
3.2 Import ETP DevKit

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



3.3 New Project

Create a new Unit Test project named ETP.Tests

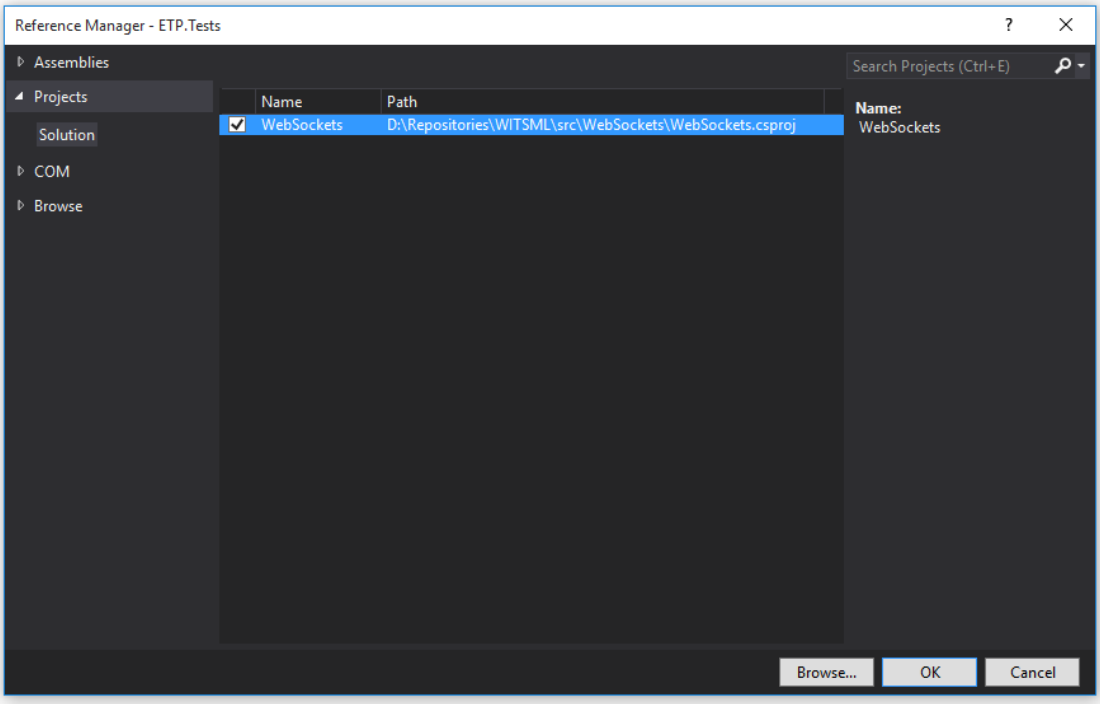


Rename the UnitTest1.cs file to EtpClientTests.cs

3.4 Update Dependencies

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

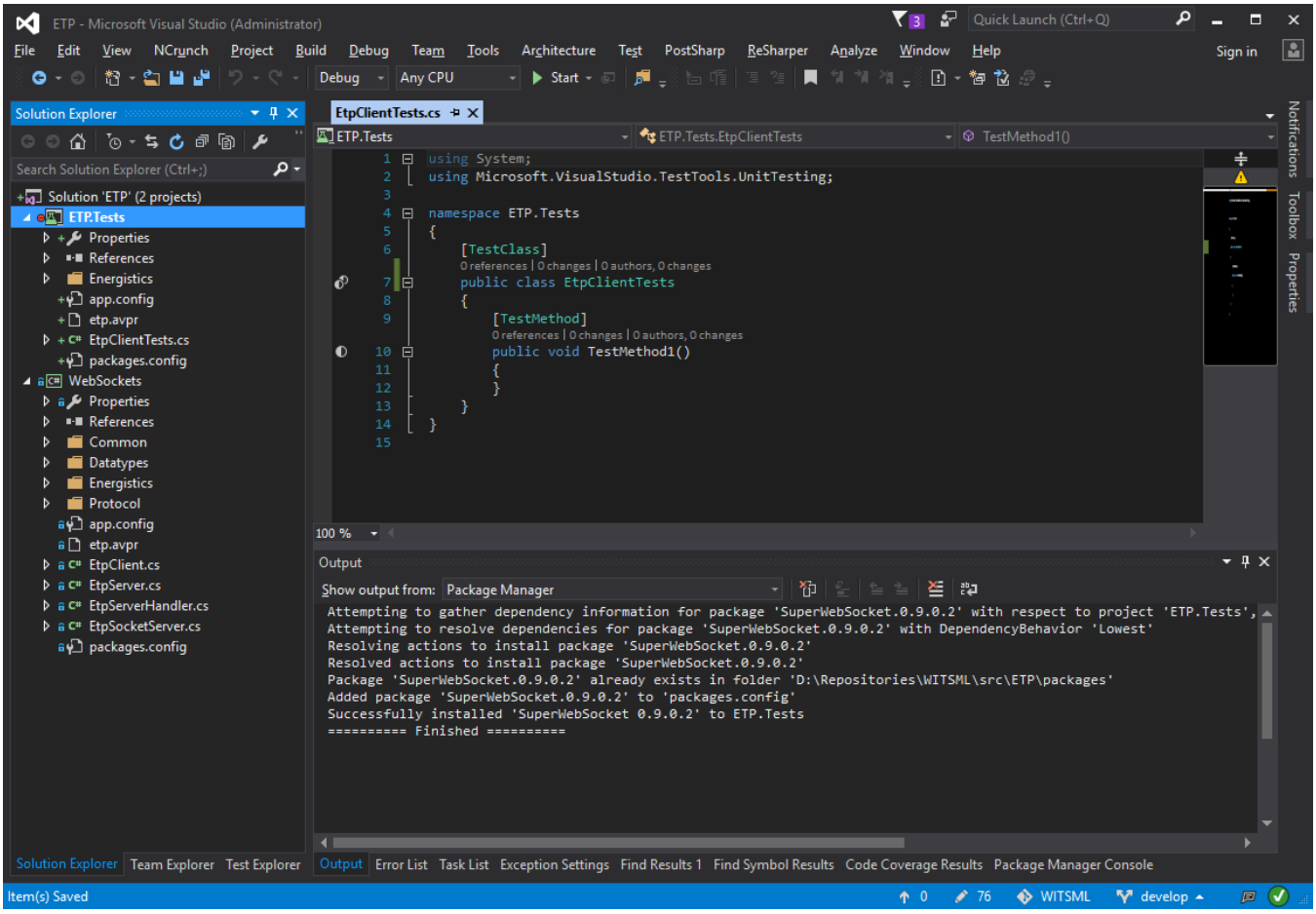
Add a project reference to the ETP.Tests project for the WebSockets library.



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 (8.0.2)
- Apache.Avro (version 1.7.7.1+)
- ETP (version 1.3.0-BUILD55) – 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 ETP Client Examples

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.

```
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.

```
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.

```
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 configurated 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;
                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;
                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());
        }
    }
}
```

4.6 Using the Discovery Protocol

5 ETP Server Examples

6 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