Serialization Training Workshop

Contents

[Introduction 2](#_Toc489210072)

[Acknowledgements 2](#_Toc489210073)

[Disclaimer 2](#_Toc489210074)

[Requirements 2](#_Toc489210075)

[Instructions 2](#_Toc489210076)

[Reminders 3](#_Toc489210077)

[Setup 3](#_Toc489210078)

[Assumptions 3](#_Toc489210079)

[Create a WebApi project 3](#_Toc489210080)

[Action 3](#_Toc489210081)

[Create a Unit Test Project 3](#_Toc489210082)

[Action 3](#_Toc489210083)

[Create an Address class 4](#_Toc489210084)

[Action 4](#_Toc489210085)

[DataContractSerializer – Add Serialization NuGet package reference 4](#_Toc489210086)

[Action 5](#_Toc489210087)

[DataContractSerializer – Create a utility class to isolate calls to the DataContractSerializer 5](#_Toc489210088)

[Action 5](#_Toc489210089)

[Overview 7](#_Toc489210090)

[DataContractSerializer – Create unit tests for the utility class 7](#_Toc489210091)

[Action 7](#_Toc489210092)

[Understanding the order of events – Simple Object 10](#_Toc489210093)

[Action 10](#_Toc489210094)

[Serialized Xml 12](#_Toc489210095)

[Overview 12](#_Toc489210096)

[Namespace 13](#_Toc489210097)

[Action 13](#_Toc489210098)

[Tracing 14](#_Toc489210099)

[Overview 14](#_Toc489210100)

[ASP.NET 4.5.\* uses Microsoft.AspNet.WebApi.Tracing 14](#_Toc489210101)

# Introduction

This workshop is intended to assist you in understanding the basics of serialization. You'll learn the basics of creating an XmlSerializer, DataContractSerializer, and how to use each with WebApi endpoints. You are assumed to have basic knowledge of C# and WebApi. While you'll create an ASP.NET Core WebApi, these principles apply to WCF and regular ASP.NET WebApi.

## Acknowledgements

* This workshop was developed by Andrew Hinkle and published by <http://www.danylkoweb.com>
* Social Media
  + Website: <http://www.penblade.com>
  + LinkedIn: <https://www.linkedin.com/in/penblade/>
  + Twitter @AndrewHinkle5
* GitHub
  + This workshop is available on GitHub for your review. If your code fails to compile or you don't get your expected results, then compare your code to the repository.
  + <https://github.com/penblade/Training.Serialization>

## Disclaimer

* Reference the License.md in the GitHub repository regarding the license. The author and publisher of this workshop hold no liability to any issues that may occur during your use of this material or recommended resources. As an example if you happen to corrupt your database and lose all the data within even though we are doing nothing with your SQL Server. Not liable. You somehow manage to decrypt secret correspondence between agents of the Mafia. Impressive. This course has nothing to do with cryptography, so I didn't teach you that, but that's pretty cool. Also, you really should watch your back, because they don't take to kindly for that. Not liable for that either in case you were wondering.

## Requirements

* Visual Studio 2017 Community or greater

## Instructions

* Most of this workshop will use the first level of bullets as a title or description of what we are doing followed by the steps to perform the action with notes in italics
* Sections after setup are broken into one or two section:
  + Overview: Discussion about the topic
  + Action: Steps that you are expected to perform
* Actual code is provided to check your work against and you can compare your code against the GitHub repository as well

## Reminders

* Microsoft.AspNet.WebApi.Tracing NuGet package must be installed and registered. See the steps later in the course.
* Microsoft.AspNet.WebApi.Tracing outputs the error to the Output window, but ONLY if the deserialization error was for the root element.

# Setup

## Assumptions

* You have setup Visual Studio and you know how to read C# code.
* You know the basics of WebApi. This course will not discuss any WebApi issues other than common serialization issues.

# Create a WebApi project

*Goal: Create a simple WebApi. This workshop will mostly be working with Unit Tests, however, we'll need the WebApi project at the end of the workshop.*

## Action

1. Create a new ASP.NET Core WebApi
   1. Click File > New Project
   2. Select Templates (left sidebar) > Visual C# >.NET Core > ASP.NET Core Web Application
      1. Name: Training.Serialization
      2. Location: C:\Projects\Training
      3. Solution name: Training.Serialization
      4. Check Create directory for solution
   3. Click OK
2. Update the target framework
   1. Right-click the test project > Properties > Change Target framework to .NETCoreApp 1.0
      1. As of 06/25/2017 .NETCoreApp 1.1 did not support DataContractSerializer.

# Create a Unit Test Project

*Goal: Create a unit test project to house all of our unit tests. We'll be working from here through most of the workshop.*

## Action

1. Create a test project with all required references to the project we are testing
   1. Right-click the solution > Add > New Project > Visual C# > .NET Core > Unit Test Project > Click Unit Test Project (.NET Core)
      1. Info: The .NET framework listed at the top says 4.5.2, however, as long as you choose the .NET Core projects, it'll update to the correct Target framework (in this workshop .NETCoreApp 1.1)
   2. Name: Training.Serialization.Tests
   3. Location: C:\Projects\Training.Serialization\Training. Serialization \Training. Serialization.Tests
   4. Click OK
   5. Right-Click the test project > Add Reference > Select Projects > Solution > Check the project "Training.Serialization > Click OK
2. Update the target framework
   1. Right-click the test project > Properties > Change Target framework to .NETCoreApp 1.0
      1. As of 06/25/2017 .NETCoreApp 1.1 did not support DataContractSerializer.
3. Rename UnitTest1.cs to Sandbox.cs and click Yes to rename the class
   1. Use this class for any tests you would like to perform without messing up the workshop tests.

# Create an Address class

*Goal: Create a simple Address object in the WebApi project.*

## Action

1. Under Training.Serialization create a Models folder
   1. Right-Click Training.Serialization.Tests > Add > Folder > Rename to Models
2. Under the Models folder create an Address class
   1. Right-Click Training.Serialization.Tests.Models > Add > Class > PrimaryAddress.cs
3. Update PrimaryAddress as follows



# DataContractSerializer – Add Serialization NuGet package reference

*Goal: Add the required NuGet package reference.*

## Action

1. Edit the Test project file
   1. Right-click Training.Serialization.Tests > Edit Training.Serialization.Tests.csproj
2. Add the PackageReference for System.Runtime.Serialization.Xml



# DataContractSerializer – Create a utility class to isolate calls to the DataContractSerializer

*Goal: Centralize the DataContractSerializer methods in a utility class since it will always be used the same for our tests.*

## Action

1. Under Training.Serialization.Tests create a DataContractSerializerUtility class
   1. Right-Click Training.Serialization.Tests > Add > Class > DataContractSerializerUtility.cs
   2. Update the class with the following logic to create the Serialize and Deserialize methods



## Overview

The DataContractSerializer is the default serializer for WCF and WebApi applications. It performs better than the XmlSerializer because it enforces some limitations such as no attributes (other than for serialization) and is in alphabetical order starting with base classes or by following the order in the XSD if it is provided. This is important if you are creating Fiddler or SoapUI tests or building your XML from scratch. There are many sites available that go into more detail on this subject.

The best way to avoid issues with maintaining your raw XML examples for Fiddler and SoapUI tests is to create a unit test that creates the objects and then uses the appropriate serializer to serialize the objects to XML. This usually helps you quickly identify aspects of the XML you would not think about such namespaces and attributes.

# DataContractSerializer – Create unit tests for the utility class

*Goal: Add unit tests that test the serialize and deserialize features.*

## Action

1. Create SerializeAndDeserializePrimaryAddress test class
   1. Right-Click Training.Serialization.Tests > Add > Class > SerializeAndDeserializePrimaryAddress.cs
   2. Add the following unit tests
      1. *By having each Assert as its own method call gives us the extra visibility we'll need to quickly identify the specific fields that did not serialize or deserialize properly.*



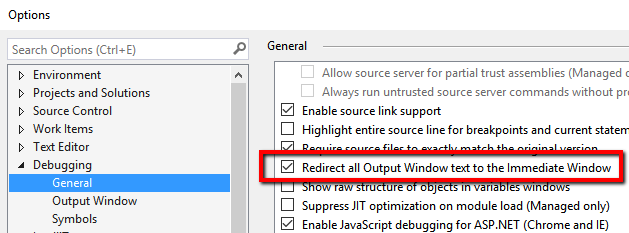


# Understanding the order of events – Simple Object

*Goal: Add logging to the serialization events to illustrate the order of events. Choose to output the logs to the Output or Immediate Window. The Immediate Window is cleaner, but either is fine.*

## Action

1. Enable Debug.WriteLine to output to the Output Window (uncheck - default) or to the Immediate Window (check)
   1. Tools > Options > Debugging > General > Redirect all Output Window text to the Immediate Window
      1. Output Window (uncheck - default)
      2. Immediate Window (check)



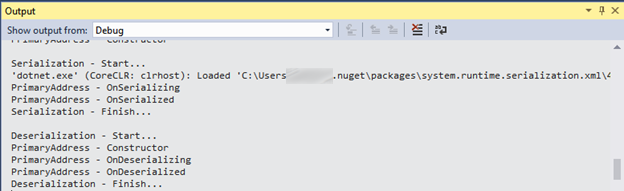
1. Add the serialization and deserialization hooks.
   1. The attributes are required. The method you assign the attribute can be named anything you want.
   2. Serialization hooks
      1. [OnSerializing]
      2. [OnSerialization]
   3. Deserialization hooks.
      1. Constructor
         1. The Constructor cannot have any parameters.
         2. Can be any scope (public, internal, protected, and private).
      2. [OnDeserializing]
      3. [OnDeserialization]
2. Add Debug.WriteLine to the serialization and deserialization hooks.



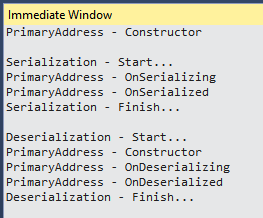
1. Add Debug.WriteLine to the TestInitialize method so we'll know when each part of the process starts and ends.



1. View the output
   1. Serialization
      1. OnSerializing
      2. OnSerialized
   2. Deserialization
      1. Constructor
      2. OnDeserializing
      3. OnDeserialized
   3. Example using Output Window.



1. Example using Immediate Window.



# Serialized Xml

## Overview

1. Namespace "xmlns" defaults to "http://schemas.datacontract.org/2004/07/{Project}.{path to object".
   1. xmlns="http://schemas.datacontract.org/2004/07/Training.Serialization.Models"
2. Namespace "xmlns:i" defaults to "http://www.w3.org/2001/XMLSchema-instance".
   1. We will be using this namespace to assist with derived types later.
3. The rest is standard XML.



# Namespace

*Goal: Define our own namespace and which fields should be included.*

## Action

1. Update PrimaryAddress with namespace: http://training.serialization.companyname.com
   1. Companies tend to name the namespace with their own domain name such as "http://{project}.{domain}.com". This is by no means a standard, just an example.



1. Run your test and note that your serialization test failed and returned an error.



1. The serialization created the xml follows without any properties, so the deserialization defaulted the values to NULL.
   1. Warning: The deserialization did not throw an exception and it clearly did not do what you wanted. You are responsible for validating values were deserialized properly and returning any errors or warnings if you decide to default if values did not deserialize.



# Tracing

*TODO: How does .NET Core handle serialization tracing?*

## Overview

*If the root element fails to deserialize, then the tracing tool outputs the error to the Visual Studio Output panel. Typically you will not want this enabled in production, so wrap it with* [*#If Debug*](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/preprocessor-directives/preprocessor-if) *or uninstall the NuGet package.*

### ASP.NET 4.5.\* uses Microsoft.AspNet.WebApi.Tracing

*If you are using the "ASP.Net 4.5.\*" framework, then you would instead install the Microsoft.AspNet.WebApi.Tracing NuGet package and register the tracing method.*

1. Click Tools > NuGet Package Manager > Manage NuGet Packages for Solution
   1. Search for Microsoft.AspNet.WebApi.Tracing
   2. Select Microsoft.AspNet.WebApi.Tracing
   3. Check your project: Training.Serialization
   4. Click Install
2. Update the WebApiConfig.Register method:
   1. config.EnableSystemDiagnosticsTracing();

