# WCF “Adopt a Dog” Service Application & Shared Library with Custom Serialization & Client "Tester" Windows Form Application

## About

This project presents a simple, but fun “Adopt a Dog” Service Application. The service provides a factory creational design pattern to create and return a single Dog object (complete with a cute photo). It is hosted using IIS Express to quickly demo and test the service with a client.

The service interface is defined not in the service application but in a *Shared Library*. The Shared Library also defines the Dog class object with custom serialization using the ISerializable interface implemented on the Dog class. This project is intended to demo custom serialization and a shared library concept between both the service and the client. With the Shared Library concept of both the service interface definition and the Dog class, there is no need to use SVCUTIL or the Service Wizard to create a tightly coupled service reference. Instead, the client uses a simple ChannelFactory concept to build a channel to the service knowing the interface details in the Shared Library. Also, it can understand and work with the Dog objects as specified in the Shared Library.

A client “tester” windows form application tests the service and provides output to the user in a simple GUI.

In addition, a short discussion and code demo of debugging service errors and activity using diagnostics and a trace listener pattern is included in this article. Some of my lessons learned in debugging service errors and custom serialization may help you on your next project.

## Architecture

The demo project consists of these component topics:

* Adopt A Dog WCF Service Application Project “WcfServiceApplication”
  + AdoptADogService (Code that Implements the Service Interface)
  + Web.config (Configuration for Trace Diagnostics on Service)
  + *Reference to the Shared Class Library*
* Shared Class Library Project “SharedLibrary”
  + IAdoptADogService (Interface for Service)
  + Dog class (defines a Dog object and Custom Serialization)
  + Photos of the Dogs
  + Logs for Trace Diagnostics
* Client “Tester to Service” Windows Form Application Project “AdoptADogClient”
  + *Reference to the Shared Class Library*
  + Main Form GUI User Interface
  + Form Code – Processes GUI User Interface

## Adopt a Dog Service Application

A WCF Service Application project was added to my Visual Studio solution. The code is available on GitHub here [].

### AdoptADogService (Code that Implements the Service Interface)

The service implementation code is a creational factory pattern that accepts parameters and returns a built Dog object to the client. The service references a shared class library called “SharedLibrary” that contains a class definition for the Dog object, including a constructor, and the interface definition. The code is available on GitHub here [].

[insert code here]

### Web.config Trace Diagnostics Configuration Settings

The service web configuration settings file is included for reference to help debug and record both activity and errors on the service to aid in testing and development. The code is available on GitHub here [].

Some Helpful Hints About Service Model Debugging

* The web configuration must be setup properly to record messages
* Some of the default parameters need to be adjusted (ex: max) to larger sizes
* Diagnostic customizations are under the <system.serviceModel> tag
* Diagnostic Trace Listeners are children of <configuration>
* A Text Listener is easier on the eye than an XML listener
* Data will be written to a file location that you must specify
* Most of the issues to listen will be on the System.ServiceModel

[insert code here]

## Shared Class Library

A Class Library project was added to my Visual Studio solution. The code is available on GitHub here [].

### IAdoptADogService (Interface for Service)

The ServiceContract for the Adopt A Dog Service allows for one possible option: make single Dog entity. The OperationContract will return a single Dog object, representing a factory created Dog object with the name, breed, age, and gender that was requested by the client. The code is available on GitHub here [].

[insert code here]

### Dog Class Hierarchy

The Dog class has the main components:

* Class Level Attributes
  + [Serializable]
  + [KnownType(typeof(DogBreed))]
  + [KnownType(typeof(GenderType))]
  + [KnownType(typeof(Bitmap))]
* Properties
  + Name (string)
  + Breed (DogBreed)
  + Gender (GenderType)
  + Age (int)
  + Photo (Bitmap)
* Constructors
  + Default
  + Parameterized
* ISerializable Implementation
  + Custom Constructor for Deserialization
  + Serializer
  + Serialization Callbacks

In addition, there are two public Enums in the class file but not in the Dog class.

* DogBreed
* GenderType

The code is available on GitHub here [].

### Class Level Attributes

The class level attributes are required for the custom serialization of the Dog class. A dog object is also composed of (contains) three different types that need to be specified for the serialization: Bitmap, DogBreed (Enum), and GenderType (Enum) all of which uses basic Serialization.

[insert code here]

### Properties

The Dog class has a few automatic properties to define a Dog object for adoption: Name (string), Breed (enum), Gender (enum), Age (int), and Photo (Bitmap).

[insert code here]

### Constructors

The Dog class has both default constructor and a parametrized constructor. The default constructor calls the parameterized constructor with default values.

[insert code here]

### ISerializable Custom Serialization and Deserialization

The class implements the ISerializable interface and has additional callbacks for process. Learn about the Microsoft technical implementation of serialization here [<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/>]

#### Serialization

The “GetObjectData” specifies the custom serialization of an object. It follows the pattern of adding properties to a Data Dictionary Serialization info object:

info.AddValue("Name of Object (Property)", PropertyName, typeof(Property Type));

[insert code here]

#### Deserialization

The constructor for deserialization is

public Dog(SerializationInfo info, StreamingContext context)

and follows the pattern of retrieving objects from the Data Dictionary Serialization info object and placing them into memory:

PropertyName = (Property Type) info.GetValue("Name of Object (Property)", typeof(Property Type));

[insert code here]

#### Callbacks

In addition to the required serialization and deserialization methods that implement ISerializable interface, the developer can include additional callback methods that get called and processed during the serialization and deserialization process.

[insert code here]

## Client “Tester to Service” Windows Form Application

The client “tester to service” is a simple windows form application project in the same solution that connects to the “Adopt A Dog Service” by use of a ChannelFactory proxy to a specified service name that is defined in the client application configuration. The client program will use this proxy to test the OperationContract or method available in the ServiceContract and return the results to the user on the form window.

The client application shares and references the “Shared Library” – a class library project. This allows the client to know how to deserialize (construct) and utilize the Dog object it receives. Also, it has access to the Adopt a Dog service interface to know exactly how it consumes the service with the help of a ChannelFactory proxy (discussed in the next section).

The user enters in dog details in the form window, then selects “Adopt A Dog” button. The form will validate the user input and then pass the data to the service to receive a newly created Dog object in the list. They can then view the dog details by selecting the dog name in the list. Multiple requests will populate additional dogs in the form list.

After the service builds and sends a single Dog object back to the client, the windows form GUI will update and add the name of the newly received Dog object to the form list. The user can then select on the individual Dog name and it will populate the Dog details in the entry boxes above and change the photo to the correct breed of dog. Note: The Photo is a bitmap that is stored on the shared class library and transmitted in the Serialization process (from service to client) for the Dog Photo property. The code is available on GitHub here [].

### ChannelFactory Proxy and Application Configuration

The application configuration file (App.config) specifies the service endpoint address, binding, and contract information. The contract is in the Shared Class Library that has the service interface. Helpful Tip: The “maxReceivedMessageSize” was set to a larger value to accommodate the large file size Bitmap images that we expect to receive with our Dog objects.

The code is available on GitHub here [].

[insert code here]

### Main Program

The main program in the client “tester” windows application first connects to the Adopt A Dog Service using a ChannelFactory proxy to the service. Based on this proxy, it then can call upon and retrieve a single Dog. The Program handles the user interface events, button clicks, validation, service calls, etc. The code is available on GitHub here [].

[insert code here]

## Demo

## Code

The entire project code repository is available on GitHub here.