DontPanic.ServiceHelpers

ServiceHelpers is a collection of classes that make sworking with WCF easier. They are designed to take the pain out of working with WCF, and make working with WCF just like working with a class. You shouldn’t have to know how something is hosted, you are just programming against an interface.

To define a service you must define an interface and a service. The interface should be marked as a “ServiceContract”, which is done using the ServiceContract attribute. The “OperationContract” attribute should be applied to methods that will be called using WCF.

Interface

    [ServiceContract]

    interface IMyService

    {

        [OperationContract]

        string TestMe(string input);

    }

Service

    class MyService : IMyService

    {

        public string TestMe(string input)

        {

            return input;

        }

    }

To call the WCF service using raw WCF you would have to do something like below.

Raw WCF

var channelFactory = new ChannelFactory<IService1>(new WSHttpBinding());

var service = channelFactory.CreateChannel(new EndpointAddress("http://locan wolhost:34244/MyService.svc"));

Console.WriteLine(service.TestMe("MyService"));

channelFactory.Close();

Calling a WCF service is awkward at best. With the WCF helpers in ServiceHelpers we can simplify this process.

With our WCF helpers calling a WCF service becomes extremely easy. Instead of a bunch of awkward lines of code, it just becomes a simple call.

Calling WCF using ServiceFactory from ServiceHelpers

var factory = new ServiceFactory();

Console.WriteLine(factory.Call<IMyService>.Proxy().TestMe("MyService"));

The code above calls a service MyService, but you don’t know that. You are just programming against a contract. Also it is important to point out that we don’t know where the service is hosted. It could be hosted in process, or in IIS hosted in Azure.

# Configuring Your Service

ServiceHelpers supports hosting a WCF service in process or as an externally visible WCF service. Regardless of how you configure you client side endpoint the calling code should be the same. Calling code should always be like below.

Calling a WCF service using ServiceFactory

var factory = new ServiceFactory();

Console.WriteLine(factory.Call<IMyService>.Proxy().TestMe("MyService"));

If you configure a WCF service using app.config or web.config, that will always be obeyed over any other setting. So if you configure a WCF client endpoint like below it will be obeyed before any other configuration for that service.

Configuring WCF Client – Default WCF Style

  <system.serviceModel>

    <client>

      <endpoint address="http://localhost:34244/MyService.svc" binding="wsHttpBinding" contract="CalledHost.IMyService" name="MyService">

      </endpoint>

    </client>

  </system.serviceModel>

You may want to configure a WCF service to run in process. This configuration can be done in two ways. First you can apply an attribute to the service. For most in process services you will want to use the attribute method.

When configuring an in process service consider (you probably should) making the service implementation private. By making the implementation private developers will only be able to consume the contract using the ServiceFactory. Developers will not be able to create an instance of the service directory.

Configuring an in process WCF service contract

    [ServiceContract]

    [InProc(typeof(MyService))]

    public interface IMyService

    {

        [OperationContract]

        string TestMe(string input);

    }

If applying an attribute to the contract cannot be done you can load the contract from web.config / app.config. You should use this configuration pattern when you need to load multiple versions of a single contract. Maybe sometimes you use one implementation of the service, and at other times you use another implementation of the service. A scenario for this might be logging. You use a file based logging when running local, but you use azure queues when running in azure.

To configure a WCF in process endpoint using web.config / app.config just add configuration like below. The configuration below will load the service “MyService” when someone requests access to the contract IMyService. The factory knows to load MyService because of the app setting key IMyService\_Load. Basically it looks for a app configuration that matches the pattern <Interface Name>\_Load.

  <appSettings>

    <add key="IMyService\_Load" value="InProcHost.MyService, InProcHost"/>

  </appSettings>

# Eliminate WCF

When working with unit tests or test clients you will often want to eliminate WCF from the picture. Doing so is incredibly easy, just add an app setting.

  <appSettings>

    <add key="DontPanic.ServiceHelper.DisableInProc" value="true"/>

  </appSettings>

# Consuming ServiceHelpers within your project.

All of your WCF service classes should derive from ServiceBase. ServiceBase provides a Factory property that you should use for all WCF calls. An example of using that is presented below. In the example below the Factory property is used to handle the WCF call to the IProductAccessor.

        public string TestMe(string input)

        {

return Factory.Proxy<IProductAccessor>().TestMe("hi");

        }

When using this within your project you probably should create your own base service class that all of your services derive from. That will give you better control over the base service. It will give you the ability to change the behavior of your services without having to change all of your services.

# The Old Lambda Way

We used to recommend calling WCF services thru Lambda expressions. With the new Proxy method this is no longer the prescribed way.

var factory = new ServiceFactory();

factory.Call<IMyService>(proxy =>

{

Console.WriteLine(proxy.TestMe("MyService"));

});