Windows Communication Foundation
An Introduction to Microsoft’s Web Service model
# Table of Contents

Windows Communication Foundation ........................................................................................................ 1

An Introduction to Microsoft’s Web Service model .............................................................................. 1

  Introduction ........................................................................................................................................ 1

Windows Communication Foundation: What is it? .............................................................................. 1

Why use Windows Communication Foundation? .................................................................................. 3

WCF at a conceptual level ....................................................................................................................... 3

WCF at an Architectural Level ................................................................................................................ 4

WCF: Application Structure and Code Examples .................................................................................. 5

  Defining a WCF Service Contract ........................................................................................................ 6

  Implementing the WCF Service Contract .............................................................................................. 7

  Hosting and Running the Service .......................................................................................................... 8

  Creating a client to consume the hosted Service .................................................................................. 9

  Using the client with the service .......................................................................................................... 10

In Conclusion ........................................................................................................................................... 11
Introduction

Windows Communication Foundation (or WCF for short) is billed as Microsoft's unified programming model for building service-oriented applications. WCF is a C Sharp and Visual Basic library used to implement web services for programming applications. WCF is a part of Microsoft's .NET framework, introduced in .NET 3.0. This framework differs from other web service frameworks in its implementation structure and reliance on the .NET 4.0 framework. It is the goal of this report to provide the reader with basic understanding of what WCF is, why a programmer would choose to use WCF, and the underlying details of its implementation.

Windows Communication Foundation: What is it?

Windows Communication Foundation is a framework for publishing web services. The purpose of specifying web services is to lighten processing load on one application, as well as provide a service for other clients or applications. Take for example, a business application containing three basic components:

1. A presentation tier for the user view, such as a shopping application
2. A server with business code, for securely processing business requests.
3. A server with data access code.
One might wonder why do we separate out these different functionalities over different machines? Why not simply have all of these functionalities run on a single machine? The answer: scalability. As a program adds more and more features, it takes more processing power on the host machine to run it. This is a big reason why, as operating systems evolve, their minimum hardware requirements increase; more new features means more required processing power. Adding more features to a program that only serves to make the program run slower is a great way to lose users.

Scalability is probably the biggest reason one should use WCF, or indeed any web service library. Another big reason we might want to build these kinds of distributed applications is to provide this enterprise service to other enterprises. PayPal is one of the more well-known examples of such a service. Nearly every popular business uses PayPal, and all PayPal requests go through PayPal’s servers, ensuring that secure payments can be made, without negatively affecting the host application performance-wise.
Why use Windows Communication Foundation?

Say we have two clients, and we need to implement a similar web service.

1. The first client is a Java application that wants its service messages to be in XML format, and the connection protocol to be HTTP.
2. The second client uses .NET, so for better performance, this client wants the messages formatted in binary and hosted over TCP protocol.

Without WCF, to satisfy the first requirement, an ASMX web service is implemented to host for the Java client. The second requirement requires a remoting service. The implementation code for both of these services are identical, but they must use different technologies, meaning that the service developer must learn two different technologies to fulfill the same service requirement for two different clients. Clearly, this is not the most scalable option. With more and more clients being written, there might be several different web services to configure.

Now we consider an implementation with WCF. WCF differs from other web services in that one service can specify multiple endpoints, each with different connection protocols and messaging formats. This means that one need only implement the initial WCF service, and then configure 2 end points, one for each client. In the endpoint configuration, the developer can specify the protocols and message formats to use. The only stipulation for this implementation is that, since .NET remoting services are not interoperable, this remoting service can only be consumed by a .NET application.

WCF at a conceptual level

Now that there is some context for what WCF is and why one might want to use it, it’s time to take a more in-depth look at how WCF services are developed and consumed. In order to facilitate the rest of this broad overview, certain terms and ideas must be made clear to the reader in order to explain WCF and its concepts properly.

WCF is based on the notion of message-based communication, and anything that can be modeled as a message (i.e. an HTTP request or Message Queuing (MSMQ) message) can be represented in a uniform way to the programming model. This allows for a unified
API over different transporting mechanism. The model distinguishes between clients, applications that initiate communication, and services, applications that wait for clients to communicate with them and then respond to that communication. A single application can act as both a client and a service.

Messages are sent between endpoints. Endpoints are places where messages are sent and/or received, and they define all the information required for message exchange. A service exposes one or more application endpoint and the client generates an endpoint that is compatible with one of the service’s endpoints. An endpoint describes a standard-based way where messages should be sent, and what the messages should look like. Services can expose this information as metadata that clients can process.

WCF at an Architectural Level

Contracts define aspects of a WCF message system. The data contract describes parameters making up every message that a service can create or consume. The service contract specifies what operations are available to clients consuming the service. This contract system allows WCF code to be loosely coupled; any implementation-level changes to the service code will not affect client-side code in such a way that the client-side code will need to be rewritten. The Service contract specifies exactly what methods are available to be used, yet obscures the implementation details, allowing for less hard-coding, and thus more scalability.
The separation of contract and actual service runtime is the key to the loosely coupled model of WCF. While the service runtime implementation may change, a client will always be able to make proper calls to the service, as the service’s useable methods are defined in the service contract; the client need not concern itself with implementation-level details.

The messaging layer is composed of channels. A channel is a component that processes a message in some way, for example, by authenticating a message. Channel sets are also known as a channel stack. Channels are responsible for operating on messages and message headers, whereas the service runtime layer concerns itself with processing message body content. Transport channels read and write messages from the network. Some examples of transports are HTTP, named pipes, and TCP. Protocol channels implement message processing, often by reading or writing additional headers to the message. Such protocols include WS-Security and WS-Reliability.

Finally, a service is hosted, and is an actual program. Like other programs, services are run in an executable. This is known as a self-hosted service. Services can also be hosted, or run in an executable managed by an external agent, such as IIS or Windows Activation Service (WAS). When the service is hosted, clients can consume the service. The messages exchanged between the client and service happen asynchronously, allowing for a parallel execution implementation.

**WCF: Application Structure and Code Examples**

Implementing a WCF Service and utilizing that service requires a few steps to set up. This code example will summarize the Getting Started Tutorial offered on Microsoft’s WCF page. This example will separate the different components of setting up and hosting a WCF service into steps.

1. Defining a WCF Service Contract
2. Implementing the Service Contract
3. Hosting and running a service
4. Creating a client to consume the service
5. Using the client with the service.
Defining a WCF Service Contract

Using Visual Studio, we first start by creating a new WCF Library Solution. For the sake of time, we will not consider Visual Basic code, and instead opt to focus on C Sharp code. Doing so creates 3 files:

1. IService1.cs, which will serve as our service contract for clients to use in code
2. Service1.cs, which will serve as the actual implementation of our service contract code
3. App.config, which serves as the contract for how clients may communicate with our service. It is here that we specify the endpoint(s) that our service will listen on.

Now we replace the generated IService1.cs code with the code for the Getting Started Library. This code specifies a simple calculator service.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace GettingStartedLib
{
    [ServiceContract(Namespace = "http://Microsoft.ServiceModel.Samples")]
    public interface ICalculator
    {
        [OperationContract]
        double Add(double x, double y);
        [OperationContract]
        double Subtract(double x, double y);
        [OperationContract]
        double Multiply(double x, double y);
        [OperationContract]
        double Divide(double x, double y);
        [OperationContract]
        double Mean(double[] nums);
    }
}
```

IService1.cs, our Service Contract

The important parts of this code are:
1. The “using System.ServiceModel;” import. This allows specification that this file is to follow the WCF service model structure.
2. The “[ServiceContract(Namespace = ...)]” line, specifying the namespace of this service, and that it is the implementation of a service contract to be used by clients.
3. The “[OperationContract]” tags above each method. Each method prefaced with this block will be useable to the client application that consumes it.

Implementing the WCF Service Contract

Now that the WCF Service contract is defined, the next task is to actually implement the code for the interface. Defining the Service Contract gives Clients the knowledge of a Service’s functions. Now all that need be done is to actually provide an implementation of this interface. The code screenshot on the next page (while not a complete implementation, for the sake of space) shows how relatively simple this is achieved. Simply importing the System.ServiceModel library and implementing the ICalculator interface is enough.
Hosting and Running the Service

Now that we’ve defined the Library to be hosted, we must specify an application to run that will host the service for clients. There is some amount of setup to be done inside this application. First, the base address of the service is assigned. For the sake of simplicity, this will be hosted locally. Second, the host program must set up a ServiceHost instance that holds the Calculator service. Third, a service endpoint must be specified. For this example, our project will use WsHttpBinding, though many other communication bindings exist. Finally, this hosting service will enable metadata exchange and start the service to be consumed by clients.
namespace GettingStartedHost
{
    class Program
    {
        static void Main(string[] args)
        {
            // Step 1 Create a URI to serve as the base address.
            Uri baseAddress = new Uri("http://localhost:8000/GettingStarted/");

            // Step 2 Create a ServiceHost instance
            ServiceHost selfHost = new ServiceHost(typeof(CalculatorService), baseAddress);
            try
            {
                // Step 3 Add a service endpoint.
                selfHost.AddServiceEndpoint(typeof(ICalculator), new WSHttpBinding(), "CalculatorService");

                ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
                smb.HttpGetEnabled = true;
                selfHost.Description.Behaviors.Add(smb);

                // Step 4 Enable metadata exchange.
                selfHost.Open();
                Console.WriteLine("Press <ENTER> to terminate service.");
                Console.ReadLine();

                // Step 5 Start the service.
                selfHost.Close();
            }
            catch (CommunicationException ce)
            {
                Console.WriteLine("An exception occurred: {0}", ce.Message);
                selfHost.Abort();
            }
        }
    }
}

A host program for our Calculator

The most important aspects of this code lie in the setup of our ServiceHost. We can verify that this code is working by running the host within Visual Studio and visiting the generated IP address that the WCF Hosting specifies. If the service is working correctly, visiting the IP address should open the service’s debug page. If there is a warning concerning metadata exchange, it’s important to check that the references in your code are updated correctly. If there is a warning concerning not having the right to host at the specified IP address, then running Visual Studio as an administrator should give the proper permissions to host the service locally.

Creating a client to consume the hosted Service

Under the same solution as the Service Host and Library, a new console application is given references to System.ServiceModel and the Calculator Service. This client must then be configured to search for the Calculator Service at its specified endpoint. The following code will be added to the Client’s App.Config file. With this configuration, the client will know where to look for the calculator service, as well as what binding (WsHttpBinding) to use to communicate with the service.
The important thing to note about changing the App.Config is the changes made to the <endpoint> tag. Specifying the proper address and associated bindings are key. Without these additions, the service won’t be able to find the service when it starts execution. Note that the address of the endpoint may change based on where the Service actually gets hosted when run. This information can be found when running the Host application from within Visual Studio.

```
<system.serviceModel>
  <bindings>
    <!-- Uses wsHttpBinding -->
    <wsHttpBinding>
      <binding name="WSHttpBinding_ICalculator" />
    </wsHttpBinding>
  </bindings>
  <!-- specifies the endpoint to use when calling the service -->
  <endpoint address="http://localhost:8000/ServiceModelSamples/Service/CalculatorService" binding="WsHttpBinding" bindingConfiguration="WSHttpBinding_ICalculator" contract="ServiceReference1.ICalculator" name="WSHttpBinding_ICalculator">
    <identity>
      <userPrincipalName value="migree@redmond.corp.microsoft.com" />
    </identity>
  </endpoint>
</system.serviceModel>
```

Using the client with the service

Now that the client is properly configured to work with the host application, it’s time to make use of the Service library. With the Host application running, a client can connect to the service and make use of the hosted service library.
using GettingStartedClient.ServiceReference1;
namespace GettingStartedClient
{
    class Program
    {
        static void Main(string[] args)
        {
            //Step 1: Create an instance of the WCF proxy.
            CalculatorClient client = new CalculatorClient();

            // Step 2: Call the service operations.
            // Call the Add service operation.
            double value1 = 100.00D;
            double value2 = 15.99D;
            double result = client.Add(value1, value2);
            Console.WriteLine("Add({0},{1}) = {2}", value1, value2, result);
        }
    }
}

This outlines just one usage of the Calculator Service, but the remaining functionality is similar between methods. The important part of this code is in the import of the service reference, the instantiation of the WCF proxy client, and the subsequent calls to the methods therein. These calls are made asynchronously and the WriteLine will wait on the result to return. All of the following code examples are given in C#, but WCF Services can be used with many languages. Indeed, any programming language capable of sending HTTP requests is capable of using WCF REST, any programming language capable of consuming web services is capable of using WCF basicHttpBinding. Possibly the greatest strength of the WCF library is the ability to host for a multitude of platforms.

In Conclusion

Windows Communication Foundation is a useful tool for specifying flexible web services. Using WCF well can increase an application's ability to scale by separating functionalities among multiple machines. Additionally, specifying multiple endpoints allows for a lot of options for customization and optimization between client types. Some of the downsides to using WCF include the somewhat daunting challenge of configuration before release. With the amount of customizability offered by WCF, it can seem a little bit overwhelming to developers new to the Library and .NET as a whole.
Bibliography and Special Thanks


Microsoft Code examples, WCF General, Overviews

https://www.youtube.com/watch?v=QmfPmqMk9Xs

Pragim Tech; WCF Usefulness, WCF Conceptualization

http://stackoverflow.com/

StackOverflow; WCF general use, code fixes, WCF usage mentality