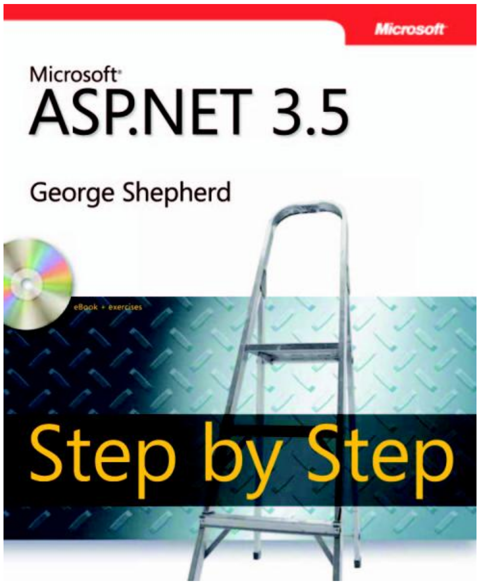


Microsoft® ASP.NET 3.5 Step by Step

George Shepherd

To learn more about this book, visit Microsoft Learning at http://www.microsoft.com/MSPress/books/11239.aspx.

9780735624269

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2007942085

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9   QWT   3 2 1 0 9 8

Distributed in Canada by H.B. Fenn and Company Ltd.

A CIP catalogue record for this book is available from the British Library.

Acquisitions Editor: Ben Ryan
Developmental Editor: Devon Musgrave
Project Editor: Kathleen Atkins
Editorial Production: P.M. Gordon Associates
Technical Reviewer: Kenn Scribner; Technical Review services provided by Content Master, a member of CM Group, Ltd.
Cover: Tom Draper Design

Body Part No. X14-40155

Table of Contents

Chapter 21
Windows Communication Foundation

After completing this chapter you will be able to

- Understand the motivation behind Windows Communication Foundation
- Understand the WCF architecture
- Implement a WCF-based server
- Build a client to use the WCF server

Distributed Computing Redux

The Windows Communication Foundation (WCF) represents one of three main pillars of .NET 3.x. These three specific highly leverageable technologies include Windows Workflow Foundation, Windows Presentation Foundation, and Windows Communication Foundation. Each of these technologies redefines programming within a certain idiom. Windows Workflow Foundation unifies the business work flow model. Windows Presentation Foundation redefines writing user interfaces, whether for Windows desktop applications or for the Web (using Silverlight). Finally, Windows Communication Foundation unifies the distributed programming model for the Microsoft platform. Clearly unifying these fragmented programming models is the main theme of .NET 3.5.

To get an idea of how fragmented the distributed computing solutions are, think back to the earliest ways to connect two computers together. At one point, the only thing you could program in any standard way was the old venerable RS232 serial connection or through a modem. Over the years, distributed computing on the Microsoft platform has grown to encompass many different protocols. For example, Windows NT supported a Remote Procedure Call mechanism that was eventually wrapped using the Distributed Component Object Model (DCOM). In addition, Windows also supports sockets programming. Near the turn of the century, Microsoft released Microsoft Message Queue (MSMQ) to support disconnected queuing-style distributed application. When it became apparent that DCOM was running into some dead ends, Microsoft introduced .NET remoting. (The “dead ends” that DCOM implemented are mainly its requirement to periodically contact client objects to remain assured of a connection, limiting scalability, its complex programming model, difficult configuration needs, and Internet-vicious security architecture.) Finally, to help supplement a wider reach available for distributed programming, Microsoft introduced an XML Web Service framework within ASP.NET (the ASMX files you looked at earlier in Chapter 20).

A Fragmented Communications API

Each of the older technologies mentioned previously has its own specific advantages—especially when you take into account the periods during computing history that they were introduced. However, having so many different means of writing distributed computing applications has led to a fragmented application programming interface (API). Making the decision as to which technology to use has always been an early decision. Earlier distributed technologies often tied your application to a specific transport protocol. If you made the wrong architectural decision or simply wanted to later migrate to a newer technology, it was often difficult if not nearly impossible to do so. Even if it could be done, it was usually an expensive proposition in terms of application redevelopment and end-user acceptance and deployment.

There are a number of programming and configuration issues involved when relying on these older technologies. The previous connection technologies coupled multiple auxiliary factors not required directly for communicating data with the communication process itself. For example, earlier distributed computing systems forced decisions such as how to format data into the early stages of design, as well as into the implementation of a distributed system. Referring back to DCOM, making DCOM remote procedure calls required an application to be tied to the DCOM connection protocol and wire format. This forced administrators to open port 135, the DCOM object discovery port, leading to immense security risks. .NET improved on things by allowing you the choice of transports and wire formats (out of the box you get a choice of using HTTP or TCP as the connection protocol, and you may use either SOAP or the .NET binary format as the wire format). However, even with those choices provided by .NET remoting, applications using classic .NET remoting are often fated to use a single connection protocol and wire format once the configuration is set. You can swap out connection protocols and wire formats, but it's not very easy.

In addition to tying wire formats and connection protocols to the implementation of a distributed system, there are many more issues cropping up when you try to connect two computers together. The minute you try to do something useful, you have to begin thinking about issues such as transactions, security, reliability, and serialization—and these issues inevitably become embedded in the application code (instead of being added later as necessary). In addition, previous communication technologies don't lend themselves to the currently in vogue Service-Oriented Architectures (SOA) where interoperability is key, although in practice interoperability is tricky to achieve.

WCF for Connected Systems

WCF's main job is to replace the previously fragmented Windows communication APIs under a single umbrella. At the same time, WCF aims to decouple the processing of communicating over a distributed system distinct from the applications themselves. When working with

WCF, you'll see that the distinctions between contracts, transports, and implementation are enforced, rather than just being a good idea. In addition, Microsoft has always been attuned to the needs of existing applications and therefore has designed WCF to accommodate partial or complete migrations from earlier communication technologies (.NET remoting or XML Web Services) to WCF-based computing.

SOA is becoming an important design influence within modern software. SOA is an architectural philosophy that encourages building large distributed systems from loosely coupled endpoints that expose their capabilities through well-known interfaces. WCF adheres to standard SOA principles, such as setting explicit boundaries between autonomous services, having services be contract and policy based (rather than being class based), having business processes be the focal point of the services (rather than services themselves), and accommodating fluctuating business models easily. WCF is designed for both high performance and maximum interoperability.

WCF represents a communication *layer*, and so introduces a level of indirection between a distributable application and the means by which the application is distributed. As an independent layer, WCF makes implementing and configuring a distributed application simpler by providing a consistent interface for managing such aspects as security, reliability, concurrency, transactions, throttling (throughput limitations for some or all callers or methods), serialization, error handling, and instance management.

While WCF is very at home when communicating via XML Web Services using SOAP (a standard for many existing Web services), it may also be configured and extended to communicate using messages based on non-SOAP formats, such as custom XML formats and RSS.

WCF is smart enough to know if both endpoints are WCF-based endpoints, in which case it will use optimized wire encoding. The structures of the messages are the same—they're just encoded in binary form. WCF includes other services often required by distributed systems. For example, WCF includes built-in queued messaging.

WCF Constituent Elements

WCF is composed of a few separate elements: endpoints, channels, messages, and behaviors. Whereas earlier communication technologies tended to couple these concepts together, WCF distinguishes them as truly separate entities. Here's a rundown of the elements of WCF.

WCF Endpoints

Endpoints define the originators and recipients of WCF communications. Microsoft has come up with a clever acronym for defining endpoints: *ABC*. That is, WCF endpoints are defined by an *address*, a *binding*, and a *contract*.

Address

The address identifies the network location of the endpoint. WCF endpoints use the addressing style of the transport moving the message. WCF addressing supports using both fully qualified addresses and relative addresses. For example, a fully qualified Internet protocol address looks like the following: *http://someserver/someapp/mathservice.svc/calculator*. WCF supports relative addressing by using a base address and then a relative address. Base addresses are registered with the service, and WCF can find services relative to the base address of the service. For example, an endpoint might comprise a whole address using a base address such as *http://someserver/someapp/mathservice.svc* and a relative address of calc.

Binding

WCF bindings specify *how* messages are transmitted. Rather than being identified simply by a transport and wire format coupled together (à la DCOM), WCF bindings are composed from a stack of binding elements which at a minimum include a protocol, a transport, and an encoder.

Contract

The final element defining an endpoint is the contract. The contract specifies the primary agreement between the client and the service as to what the service can do for the client. The contract specifies the information to be exchanged during a service call.

WCF expresses a Service Contract as a .NET interface adorned with the *[ServiceContract]* attribute. Methods within the WCF contract interface are annotated with the *[OperationContract]* attribute. WCF interfaces may pass data structures as well. Data members within the structures are exposed as properties and adorned with the *[DataMember]* attribute.

Channels

WCF channels represent the message transmission system. WCF defines *protocol channels* and *transport channels*. Protocol channels add services such as security and transactions independently of transport. Transport channels manage the physical movement of bytes between endpoints (for example, WCF uses protocols such as MSMQ, HTTP, P2P, TCP, or Named Pipes). WCF uses a factory pattern to make creating channels consistent.

Behaviors

In WFC, the service contract defines *what* the service will do. The service contract implementation specifies exactly *how* the service contract functionality works. However, one of the hallmarks of a distributed system is that it usually requires some add-on functionality that may not necessarily be tied to contract implementation. For example, when securing a Web service, authenticating and authorizing the client may be necessary, but it's usually not part of the service contract. WFC implements this kind of add-on functionality through *behaviors*. Behaviors implement the SOA higher-order notion of policy and are used to customize local execution.

Behaviors are governed by attributes—the main two of which are the *ServiceBehaviorAttribute* and the *OperationBehaviorAttribute*. The *ServiceBehaviorAttribute* and *OperationBehaviorAttribute* attributes control the following aspects of the service execution:

- Impersonation
- Concurrency and synchronization support
- Transaction behavior
- Address filtering and header processing
- Serialization behavior
- Configuration behavior
- Session lifetime
- Metadata transformation
- Instance lifetimes

Applying these attributes to modify the server execution is easy. Just adorn a service or operation implementation with the appropriate attribute and set the properties. For example, to require that callers of an operation support impersonation, adorn a service operation with the *OperationBehavior* attribute and set the *Impersonation* property to *ImpersonationOption.Require*.

Messages

The final element of WCF is the actual message. WCF messages are modeled on SOAP messages. They are composed of an envelope, a header, a body, and addressing information. Of course, messages also include the information being exchanged. WCF supports three Message Exchange Patterns: one-way, request-response, and duplex. One-way messages are passed from the transmitter to the receiver only. Messages passed using the request response pattern are sent from the transmitter to the receiver, and the receiver is expected to send a reply back to the originator. Messages using the request response pattern block until the receiver sends a response to the originator. When using the duplex messaging, services can call back to the client while executing a service requested by the client. The default Message Exchange Pattern is request-response.

How WCF Plays with ASP.NET

Although WCF applications may be hosted by manually written servers, ASP.NET makes a perfectly good host. You can either write your own Windows Service to act as a host, or you can take advantage of a readily available Windows Service, IIS, and consequently ASP.NET. WCF and ASP.NET may co-exist on a single machine in two different modes—side-by-side mode and ASP.NET compatibility mode. Here's a rundown of these two modes.

Side-by-Side Mode

When running in side-by-side mode, WCF services hosted by Internet Information Services (IIS) are co-located with ASP.NET applications composed of .ASPX files and ASMX files (and ASCX and ASHX files when necessary). ASP.NET files and WCF services reside inside a single, common Application Domain (AppDomain). When run this way, ASP.NET provides common infrastructure services such as AppDomain management and dynamic compilation for both WCF and the ASP.NET HTTP runtime. WCF runs in side-by-side mode with ASP.NET by default.

When running in side-by-side mode, the ASP.NET runtime manages only ASP.NET requests. Requests meant for a WCF service go straight to the WCR-based service. Although the ASP.NET runtime does not participate in processing the requests, there are some specific ramifications of running in side-by-side mode.

First, ASP.NET and WCF services can share AppDomain state. This includes such items as static variables and public events. Although it shares an AppDomain with ASP.NET, WCF runs independently—meaning some features you may count on when working with ASP.NET become unavailable. Probably the major restriction is that there's no such thing as a current *HttpContext* from within a WCF service (despite WCF's architectural similarity to ASP.NET's runtime pipeline). Architecturally speaking, WCF can communicate over many different protocols, including but not limited to HTTP, so an HTTP-specific context may not even make sense in many given scenarios. Second, authentication and authorization can get a bit tricky.

Even though WCF applications do not interfere with ASP.NET applications, WCF applications may access various parts of the ASP.NET infrastructure such as the application data cache. In fact, this chapter's example shows one approach to accessing the cache.

ASP.NET Compatibility Mode

WCF is designed primarily to unify the programming model over a number of transports and hosting environments. However, there are times when a uniform programming model with this much flexibility is not necessary and the application may desire or even require some of the services provided by the ASP.NET runtime. For those cases, WCF introduces the ASP.NET compatibility mode. WCF's ASP.NET compatibility mode lets you run your WCF application as a full-fledged ASP.NET citizen, complete with all the functionality and services available through ASP.NET.

WCF services that run using ASP.NET compatibility mode have complete access to the ASP.NET pipeline and execute through the entire ASP.NET HTTP request life cycle. WCF includes an implementation of *IHttpHandler* that wraps WCF services and fosters them through the pipeline when run in ASP.NET compatibility mode. In effect, a WCF service running in ASP.NET compatibility mode looks, tastes, and feels just like a standard ASP.NET Web service (that is, an ASMX file).

WCF applications running under the ASP.NET compatibility mode get a current *HttpContext* with all its contents—the session state, the *Server* object, the *Response* object, and the *Request* object. WCF applications running as ASP.NET compatible applications may secure themselves by associating Windows Access Control Lists (ACLs) to the service's .svc file. In this manner, only specific Windows users could use the WCF service. ASP.NET URL authorization also works for WCF applications running as ASP.NET compatible applications. The pipeline remains arbitrarily extensible for WCF applications running as ASP.NET applications because service requests are not intercepted as with the general purpose side-by-side mode—they're managed by ASP.NET for the entire request life cycle.

You can turn on WCF's ASP.NET compatibility mode at the application level through the application's web.config file. You can also apply ASP.NET compatibility to a specific WCF service implementation.

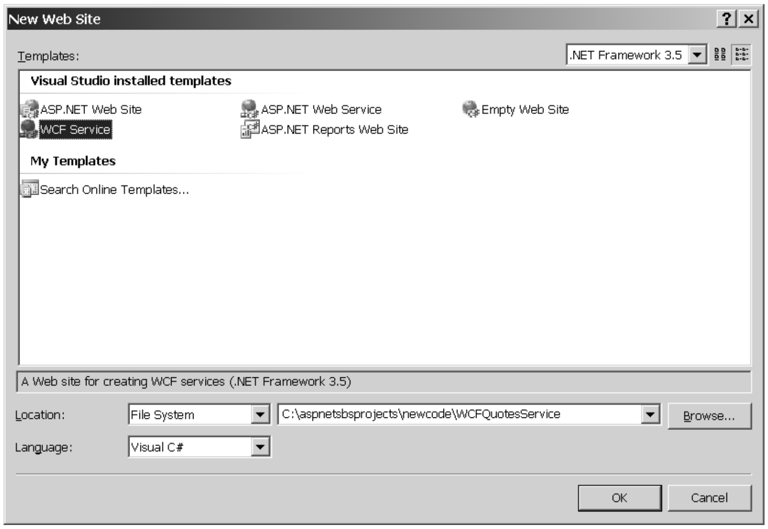
Writing a WCF Service

Here's an example of WCF service to help illustrate how WCF works. Recall the XML Web Service example application from Chapter 20, the QuoteService that doled out pithy quotes to any client wishing to invoke the service. The example here represents the same service—but using a WCF-based Web site instead of an ASMX-based Web service. This way, you'll see what it takes to write a WCF-based service and client, and you'll see some of the differences between WCF-based services and ASMX-based services (there are a couple of distinct differences).

QuotesService

1. Start by creating a WCF project. This example takes you through the nuts and bolts of developing a working WCF application that may be accessed from any client anywhere. Start Visual Studio 2008. Select **File**, **New**, **Web Site** and choose **WCF Service** from the

available templates. Name the site *WCFQuotesService*. The following graphic shows the **New Web Site** dialog box:



New Web Site

Templates: .NET Framework 3.5

Visual Studio installed templates

ASP.NET Web Site | ASP.NET Web Service | Empty Web Site

WCF Service | ASP.NET Reports Web Site

My Templates

Search Online Templates...

A Web site for creating WCF services (.NET Framework 3.5)

Location: File System C:\aspnetsbsprojects\newcode\WCFQuotesService

Language: Visual C#

2. Examine the files created by Visual Studio. The *App_Code* directory includes two files: IService.cs and Service.cs. These two files are placeholders representing the WCF contract (as a .NET interface type) and a class implementing the contract.

3. Tweak the files produced by Visual Studio. Name the code files representing the service. *IService.cs* should become *IQuotesService.cs*, and *Service.cs* should become *QuotesService.cs*.

4. Change the service interface name from *IService* to *IQuotesService* and change the service class name from *Service* to *QuotesService*. Use Visual Studio's refactoring facilities to do this. That is, highlight the identifier you want to change, click the right mouse button in the text editor and select **Rename** from the Refactoring menu. Visual Studio will make sure the change is propagated through the entire project.

5. Borrow the *QuotesCollection* object from the Web Service chapter. Bring in the QuotesCollection.cs file from the QuotesService Web site. To do this, select the *App_Code* directory in the WCFQuotesService project. Click the right mouse button and select **Add Existing Item**. Go to the Web services project and pick up the *QuotesCollection* class by bringing in the file QuotesCollection.cs. The QuotesCollection.cs file will be copied into your WCF solution and added to the project.

6. Borrow the QuotesCollection.xml and QuotesCollection.xsd from the Web service example. Select the *App_Data* directory in the WCFQuotesService project. Click the right mouse click and select **Add Existing Item**. Go to the Web services project and pick up the XML and XSD files.

7. Develop a data contract. Now that the data and the data management code are in place, the service needs a way to expose itself. It's time to develop a contract for the service. First, create a structure for passing quotes back and forth. Open the file IQuotesService.cs to add the data and operation contracts. To do so, first delete the *CompositeType* class Visual Studio placed there for you as an example. In its place, type in the following code for the *Quote* structure. The *Quote* structure should contain three strings—one to represent the quote text and separate strings to represent the originator's first and last names. Expose the strings as properties adorned with the *[DataMember]* attribute.

```
[DataContract]
public struct Quote
{
    private String _strQuote;

    [DataMember]
    public String StrQuote
    {
        get { return _strQuote; }
        set { _strQuote = value; }
    }

    private String _strOriginatorLastName;

    [DataMember]
    public String StrOriginatorLastName
    {
        get { return _strOriginatorLastName; }
        set { _strOriginatorLastName = value; }
    }

    private String _strOriginatorFirstName;

    [DataMember]
    public String StrOriginatorFirstName
    {
        get { return _strOriginatorFirstName; }
        set { _strOriginatorFirstName = value; }
    }

    public Quote(String strQuote,
                 String strOriginatorLastName,
                 String strOriginatorFirstName)
    {
        _strQuote = strQuote;
        _strOriginatorLastName = strOriginatorLastName;
        _strOriginatorFirstName = strOriginatorFirstName;
    }
}
```

8. Next, develop a service contract for the service. In the IQuotesService.cs file, update the interface to include methods to get a single quote, add a quote, and get all the quotes.

```
using System.Data; // must be added to identify DataSet

[ServiceContract]
public interface IQuotesService
{
    [OperationContract]
    Quote GetAQuote();

    [OperationContract]
    void AddQuote(Quote quote);

    [OperationContract]
    DataSet GetAllQuotes();
}
```

9. Next, implement the service contract. Much of the work for this step is already done from the Web service chapter example. However, there are a couple of critical differences between the two implementations (those being the Web service implementation and the WCF implementation). Open the file QuotesService.cs to add the implementation. Start by implementing a method that loads the quotes into memory and stores the collection and the ASP.NET cache. Although this application is an ASP.NET application, ASP.NET handles WCF method calls earlier in the pipeline than normal ASP.NET requests, and because of that there's no such thing as a current *HttpContext* object. You can still get to the cache through the *HttpRuntime* object, which is available within the context of WCF. The *HttpRuntime.AppDomainAppPath* property includes the path to the application that's useful for setting up a cache dependency for the XML file containing the quotes.

```
using System.Web; // must be added to identify HttpRuntime
using System.Web.Caching; // must be added to identify Cache
using System.Data; // must be added to identify DataSet

public class QuotesService : IQuotesService
{
    QuotesCollection LoadQuotes()
    {
        QuotesCollection quotesCollection;
        quotesCollection =
            (QuotesCollection)
            HttpRuntime.Cache["quotesCollection"];
        if (quotesCollection == null)
        {
            quotesCollection = new QuotesCollection();

            String strAppPath;
            strAppPath = HttpRuntime.AppDomainAppPath;
            String strFilePathXml =
                String.Format("{0}\\App_Data\\QuotesCollection.xml", strAppPath);
            String strFilePathSchema =
                String.Format("{0}\\App_Data\\QuotesCollection.xsd", strAppPath);

            quotesCollection.ReadXmlSchema(strFilePathSchema);
            quotesCollection.ReadXml(strFilePathXml);

            CacheDependency cacheDependency =
                new CacheDependency(strFilePathXml);

            HttpRuntime.Cache.Insert("quotesCollection",
                           quotesCollection,
                           cacheDependency,
                           Cache.NoAbsoluteExpiration,
                           Cache.NoSlidingExpiration,
                           CacheItemPriority.Default,
                            null);
        }
        return quotesCollection;
    }
// more code will go here...
}
```

10. Next, implement the *GetAQuote* operation. Call *LoadQuotes* to get the *QuotesCollection* object. Generate a random number between 0 and the number of quotes in the collection and use it to select a quote within the collection. Create an instance of the *Quote* structure and return it after populating it with the data from the stored quote.

```
public class QuotesService : IQuotesService
{
    // LoadQuotes here...

    public Quote GetAQuote()
    {
        QuotesCollection quotesCollection = this.LoadQuotes();
        int nNumQuotes = quotesCollection.Rows.Count;

        Random random = new Random();
        int nQuote = random.Next(nNumQuotes);
        DataRow dataRow = quotesCollection.Rows[nQuote];
        Quote quote = new Quote((String)dataRow["Quote"],
                                (String)dataRow["OriginatorLastName"],
                                (String)dataRow["OriginatorFirstName"]);
        return quote;
    }
    // more code will go here...
}
```

11. Implement *AddAQuote*. Call *LoadQuotes* to get the *QuotesCollection*. Create a new row in the *QuotesCollection* and populate it with information coming from the client (that is, the *Quote* parameter). Use the *HttpRuntime.AppDomainAppPath* to construct the path

to the QuotesCollection.XML file and use the *QuotesCollection*'s *WriteXml* method to re-serialize the XML file. *WriteXml* is available from the *QuotesCollection* class because *QuotesCollection* derives from *System.Data.DataTable*. Because it was loaded in the cache with a file dependency, the cache will be invalidated and the new quotes collection will be loaded the next time around.

```
public class QuotesService : IQuotesService
{
    // LoadQuotes here...
    // GetAQuote here

    public void AddQuote(Quote quote)
    {
        QuotesCollection quotesCollection = this.LoadQuotes();

        DataRow dr = quotesCollection.NewRow();
        dr[0] = quote.StrQuote;
        dr[1] = quote.StrOriginatorLastName;
        dr[2] = quote.StrOriginatorFirstName;
        quotesCollection.Rows.Add(dr);

        string strAppPath;
        strAppPath = HttpRuntime.AppDomainAppPath;

        String strFilePathXml =
            String.Format("{0}\\App_Data\\QuotesCollection.xml", strAppPath);
        String strFilePathSchema =
            String.Format("{0}\\App_Data\\QuotesCollection.xsd", strAppPath);

        quotesCollection.WriteXmlSchema(strFilePathSchema);
        quotesCollection.WriteXml(strFilePathXml);
    }
}
```

12. Finally, implement the *GetAllQuotes* operation. Create a new *DataSet*, load the quotes, and add the *QuotesCollection* to the data set as the first table. Then return the *DataSet*.

```
public class QuotesService : IQuotesService
{
    // LoadQuotes here
    // GetAQuote here
    // AddQuote here

    public DataSet GetAllQuotes()
    {
        QuotesCollection quotesCollection = LoadQuotes();
        DataSet dataSet = new DataSet();
        dataSet.Tables.Add(quotesCollection);
        return dataSet;
    }
}
```

13. Tweak the web.config file. Now that the service is implemented, the web.config file needs to be tweaked just slightly to expose the service. Visual Studio created this file and so exposes the service that it generated—the one named *Service*. However, you renamed the service to give it a more useful name in the code and the service needs to be exposed as *QuotesService* now. Update the web.config file to reflect the change. Change the *name* attribute in the service node to be *QuotesService*. Change the *contract* attribute in the endpoint node to be *IQuotesService* to match the interface name.

```
<system.serviceModel>
   <services>
      <service
         name="QuotesService"
         behaviorConfiguration="ServiceBehavior">
         <!-- Service Endpoints -->
         <endpoint address=""
           binding="wsHttpBinding"
           contract="IQuotesService"/>
         <endpoint address="mex"
           binding="mexHttpBinding"
           contract="IMetadataExchange"/>
      </service>
   </services>
   <behaviors>
      ...
   </behaviors>
</system.serviceModel>
```

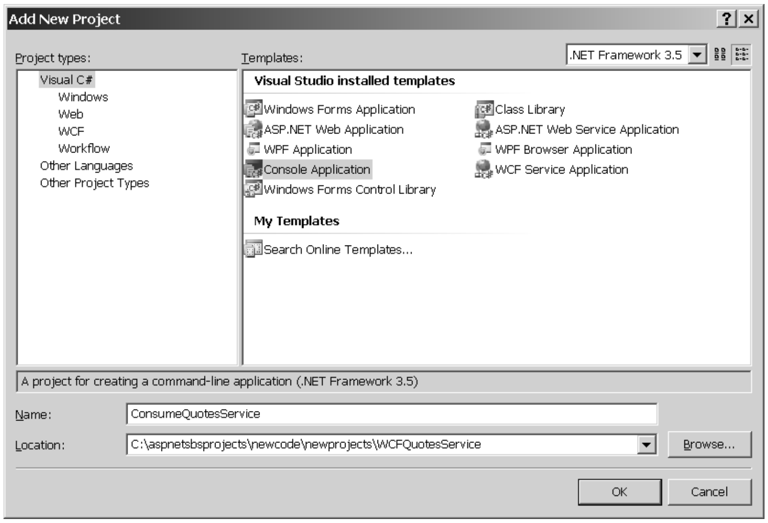
That does it for building a WCF service hosted through ASP.NET that may be called from anywhere in the world (that has Internet service available, that is). In many ways, this is very similar to writing a classic ASP.NET Web service. However, because this service runs in ASP.NET side-by-side mode, there's no such thing as a current *HttpContext* (as is available in normal ASP.NET applications). In many cases, this may not be necessary. You can get to many of the critical ASP.NET runtime services (for example, the Cache) via the *HttpRuntime* object. If you need ASP.NET's full support (such as for session state if the WCF service you write depends on session data), WCF supports the ASP.NET Compatibility mode.

Building a WCF Client

A WCF service is useless without any clients around to employ it. This section illustrates how to build a client application that consumes the Quotes service. Here, you'll see how Visual Studio makes it very easy to create a reference to a service. You'll see how to make WCF service calls both synchronously and asynchronously.

Building the QuotesService client

1. Start Visual Studio 2008. Open the QuotesService solution and add a new project to it. Make it a console application named *ConsumeQuotesService*. The following graphic illustrates adding the Console project to the solution:



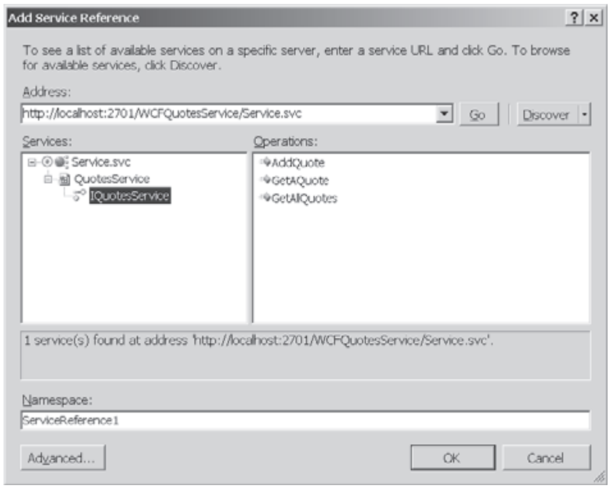
.NET Framework 3.5

A project for creating a command-line application (.NET Framework 3.5)

Name: ConsumeQuotesService

Location: C:\aspnetsbsprojects\newcode\newprojects\WCFQuotesService

2. Create a reference to the QuotesService WCF application. Click the right mouse button on the **ConsumeQuotesService Project** tree node within the solution explorer. Select **Add Service Reference**. When the **Add Service Reference** dialog box shows, click the **Discover** button. Select the Service.svc file from this project and expand its associated tree node. After a minute, the dialog will display the service contracts that are available through the service. Expand the Services tree in the left pane to make sure you see the *IQuotesService* contract. Notice the namespace given by the dialog box—*ServiceReference1*. DON'T click **OK** yet. The following graphic shows adding the Service Reference:



To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.

Address: http://localhost:2701/WCFQuotesService/Service.svc

1 service(s) found at address 'http://localhost:2701/WCFQuotesService/Service.svc'.

Namespace: ServiceReference1

3. Click the **Advanced…** button. Click on the **Generate Asynchronous Operations** radio button so that Visual Studio generates the asynchronous versions of the proxy methods.

4. Click the **OK** button to add the service reference. Visual Studio will produce a new directory within the ConsumeQuotesService project directory named *ServiceReferences*. Visual Studio generates information about the service in the form of XML files, XSD files, and a WSDL file (among others). You'll also get source code for a proxy class that will call the service on your behalf (by default, the proxy lands in a file named *Reference.cs*.

5. Try calling the *GetAQuote* operation. Calling the proxy methods generated for the WCF service calls can be a bit verbose from time to time, but they are effective and it's much better than setting everything up manually by yourself. First, create an instance of the *QuotesServiceClient*, available from the *ServiceReference* you just created. Create an instance of the *ServiceReference1.Quote* structure to receive the results of calling *GetAQuote*. Call *GetAQuote* from the *QuotesServiceClient*, and print the result on the console.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsumeQuotesService
{
    class Program
    {
        static void Main(string[] args)
        {
            // Get a single random quote
            ServiceReference1.QuotesServiceClient quotesServiceClient =
              new ServiceReference1.QuotesServiceClient();

            ServiceReference1.Quote quote = quotesServiceClient.GetAQuote();

            Console.WriteLine("Getting a single quote: " + quote.StrQuote);
            Console.WriteLine();
        }
    }
}
```

6. Now try calling *AddAQuote*. This will be very much like calling *GetAQuote*. However, this time the request requires some parameters. Create an instance of the *Quote* (available from the *ServiceReference*). Find some pithy quote somewhere (or make one up) and plug it into the *Quote* object along with the first and last names of the originator. You can use the same instance of the *QuotesServiceClient* to call *AddAQuote*, passing the *Quote* object in. The next call to *GetAllQuotes* will reveal that the quote was added to the quotes collection (which we'll see in a minute).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace ConsumeQuotesService
{
    class Program
    {
        static void Main(string[] args)
        {
            // Get a single random quote
            ...

            // Now add a quote
            ServiceReference1.Quote newQuote = new ServiceReference1.Quote();
            newQuote.StrQuote = "But to me nothing - the negative, the empty" +
                "- is exceedingly powerful.";
            newQuote.StrOriginatorFirstName = "Alan";
            newQuote.StrOriginatorLastName = "Watts";

            quotesServiceClient.AddQuote(newQuote);

            Console.WriteLine("Added a quote");
            Console.WriteLine();
        }
    }
}
```

7. Now try calling *GetAllQuotes*. By now you should know the pattern pretty well. Use the *QuotesServiceClient* to call *GetAllQuotes*. *GetAllQuotes* will return a *DataSet* object that will contain a collection of all the quotes, so declare one of those, too. Use the *QuotesServiceClient* object to call *GetAllQuotes*. When the call returns, use the *DataSet* object to print the quotes to the console. Be sure to include the *System.Data* namespace so the compiler understand the *DataSet*.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;

namespace ConsumeQuotesService
{
    class Program
    {
        static void Main(string[] args)
        {
            // Get a single random quote

            // Now add a quote

            // Now get all the quotes
            DataSet dataSet = quotesServiceClient.GetAllQuotes();
            DataTable tableQuotes = dataSet.Tables[0];
            foreach (DataRow dr in tableQuotes.Rows)
            {
                System.Console.WriteLine(dr[0] + " " +
                dr[1] + "  " + dr[2]);
            }
        }
    }
}
```

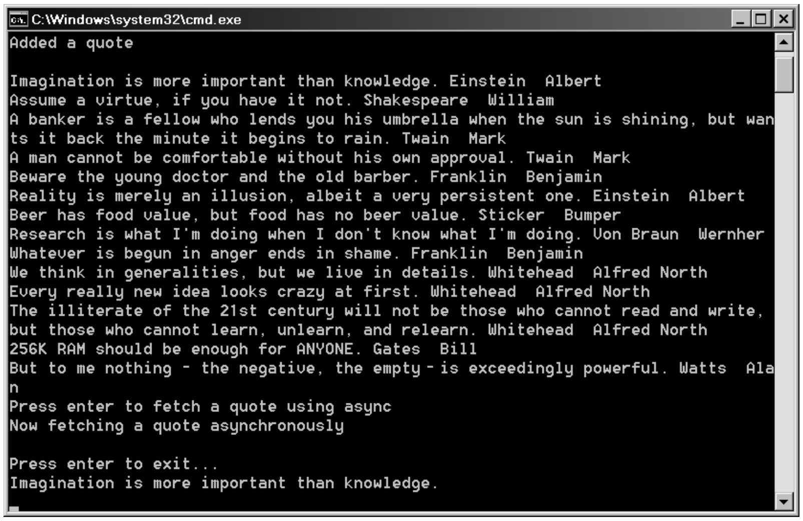
8. Try calling *GetAQuote* asynchronously. The proxy created by Visual Studio supports asynchronous invocation if you ask it to generate the asynchronous methods. To call *GetAQuote* asynchronously, you'll need to implement a callback method that WCF will call when the method is done executing. Add a static method named *GetAQuoteCallback* to your *Program* class. Have the method return void, and take *IAsyncResult* as a parameter. When WCF calls back into this method, the *IAsyncResult* parameter will be an instance of the class originating the call—an instance of *QuotesServiceClient*. Declare an instance of the *ServiceReference1.QuotesServiceClient* class and assign it by casting the *IAsyncResult* parameter's *AsyncState* property to the *ServiceReference1.QuotesServiceClient* type. Then declare an instance of the *Quote* class and harvest the quote by calling *QuotesServiceClient.EndGetAQuote*, passing the *AsyncResult* parameter. Finally, write the quote out to the console.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;

namespace ConsumeQuotesService
{
    class Program
    {
        static void Main(string[] args)
        {
            // Get a single random quote

            // Now add a quote

            // Now get all the quotes
        }

        static void GetAQuoteCallback(IAsyncResult asyncResult)
        {
            ServiceReference1.QuotesServiceClient quotesServiceClient =
                 (ServiceReference1.QuotesServiceClient)
                 asyncResult.AsyncState;
            ServiceReference1.Quote quote =
                quotesServiceClient.EndGetAQuote(asyncResult);

            Console.WriteLine(quote.StrQuote);
        }
    }
}
```

9. Now make the asynchronous call to *GetAQuote*. This is easy—just call the *QuotesServiceClient*'s *BeginGetAQuote* method from the *Program* class's *Main* method. Pass in the *GetAQuoteCallback* method you just wrote as the first parameter, and the *QuotesServiceClient* object as the second parameter. Add a call to *System.Console.ReadLine* to pause the main thread so that the asynchronous call has time to execute.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;

namespace ConsumeQuotesService
{
    class Program
    {
        static void Main(string[] args)
        {
            // Get a single random quote

            // Now add a quote

            // Now get all the quotes

            // Now call GetAQuote asynchronously
            System.Console.WriteLine(
                "Now fetching a quote asynchronously");
            Console.WriteLine();

            quotesServiceClient.BeginGetAQuote(GetAQuoteCallback,
               quotesServiceClient);

            Console.WriteLine("Press enter to exit...");
            Console.ReadLine();
        }

        static void GetAQuoteCallback(IAsyncResult asyncResult)
        {
             // implementation removed for clarity
        }
    }
}
```

10. Run the program to watch the console application call the WCFQuotesService. You should see the following output:



```
Added a quote

Imagination is more important than knowledge. Einstein  Albert
Assume a virtue, if you have it not. Shakespeare  William
A banker is a fellow who lends you his umbrella when the sun is shining, but wan
ts it back the minute it begins to rain. Twain  Mark
A man cannot be comfortable without his own approval. Twain  Mark
Beware the young doctor and the old barber. Franklin  Benjamin
Reality is merely an illusion, albeit a very persistent one. Einstein  Albert
Beer has food value, but food has no beer value. Sticker  Bumper
Research is what I'm doing when I don't know what I'm doing. Von Braun  Wernher
Whatever is begun in anger ends in shame. Franklin  Benjamin
We think in generalities, but we live in details. Whitehead  Alfred North
Every really new idea looks crazy at first. Whitehead  Alfred North
The illiterate of the 21st century will not be those who cannot read and write,
but those who cannot learn, unlearn, and relearn. Whitehead  Alfred North
256K RAM should be enough for ANYONE. Gates  Bill
But to me nothing - the negative, the empty - is exceedingly powerful. Watts  Ala
n
Press enter to fetch a quote using async
Now fetching a quote asynchronously

Press enter to exit...
Imagination is more important than knowledge.
```

Summary

Out of the box, The Windows Communication Foundation unifies the programming interface for the two modern .NET remoting technologies: standard .NET remoting and .NET XML Web Services (and will also accommodate MSMQ and sockets communication). Although effective at the time, the communication infrastructures of the late 1980s through the mid-2000s narrowed the design and implementation possibilities for a distributed system. WCF offers a single framework for creating a distributed system. WCF marks clear boundaries between the elements of a distributed system, making it much easier to design a distributed system independently of the communication mechanism it will use eventually. WCF doesn't hem you into specific communication infrastructure choices early on. In addition, WCF makes it very straightforward to add features such as security and transaction management.

Distributed WCF applications are composed of several different elements: endpoints, channels, messages, and behaviors. An endpoint is defined by an address, a binding, and a contract. Endpoints specify message originators and recipients. WCF channels represent the means by which messages are transmitted. WCF defines protocol channels and transport channels. Messages are the actual data sent between the endpoints, and behaviors specify how a WCF service operates at run time, allowing you to configure the runtime characteristics of the services, such as concurrency and security.

WCF applications are easily hosted by ASP.NET. The Visual Studio ASP.NET wizard provides a template for creating WCF applications. When hosting WCF applications via ASP.NET, you have two options: running in ASP.NET side-by-side mode and running in ASP.NET compatibility mode. When running in ASP.NET side-by-side mode, the WCF services may run in the same AppDomain and share state and event handlers exposed by other assemblies loaded in the AppDomain. However, normal ASP.NET features such as session state and the current request context are unavailable. You may get to certain ASP.NET features such as the application cache through the *HttpRuntime* class. When running under ASP.NET compatibility mode, calls to the WCF service are full-fledged ASP.NET requests. The WCF requests that run within an ASP.NET compatible service have full access to all of ASP.NET's features, including access to the current *HttpContext* and the session state.

Chapter 21 Quick Reference

To	Do This
Create a WCF-enabled Web site	In Visual Studio, choose **File**, **New**, **Web Site** and select **WCF Service** from the available templates. This will produce a WCF-enabled Web site for you and will stub out a default contract and implementation that you may change to fit your needs.
Create a service contract	Service contracts are defined as .NET interfaces. The entire interface should be adorned with the *[ServiceContract]* attribute. Interface members meant to be exposed as individual services are adorned with the *[OperationContract]* attribute. Data structures may be passed through the interface. Structure members meant to be visible through the interface are adorned with the *[DataContract]* attribute.
Implement the service contract	Create a class that derives from the interface defining the service contract and implement the members.
Expose the WCF service as an ASP.NET application	Make sure that the web.config file mentions the service contract and the implementation.
Create a client to consume the WCF service	Use the Add Service Reference menu item found in the project's context menu (exposed from Visual Studio's Solution Explorer) to discover and locate the service metadata. Alternatively, use the ServiceModel Metadata Utility Tool (packaged as an assembly named *Svcutil.exe*).
Customize the service's local execution, managing execution aspects such as security, instance lifetime, and threading	Apply the *ServiceBehaviorAttribute* and *OperationBehaviorAttribute* attributes as necessary to control the following aspects of the service execution: instance lifetimes, concurrency and synchronization support, configuration behavior, transaction behavior, serialization behavior, metadata transformation, session lifetime, address filtering and header processing, and impersonation.
Access the ASP.NET application cache from a standard WCF application (one not configured to run in ASP.NET compatibility mode)	Use the *HttpRuntime.Cache* property.