

عنوان مضمون

# Visual Programming-II

توسط : صفری

بهار 1397

# سرفصل ها

- collections, generics and Events, WCF services

windows services and WCF services

WCF services

# Windows Communication Foundation(WCF)

- A **web service** is like a website that is used by a computer instead of a person.
- For example, instead of browsing to a website about your favorite TV program, you might instead use a desktop application that pulled in the same information via a web service.
- The .NET Framework has supported web services for some time now. However, in the more recent versions of the framework, web services have been combined with another technology, called *remoting*, to create *Windows Communication Foundation* (WCF), which is a generic infrastructure for communication between applications.

# WHAT IS WCF?

- WCF is a technology that enables you to create services that you can access from other applications across process, machine, and network boundaries. You can use these services to share functionality across multiple applications, to expose data sources, or to abstract complicated processes.

# WCF CONCEPTS

- WCF communication protocols
- endpoints
  - Addresses
  - bindings
  - Contracts
- Message patterns
- Behaviors
- Hosting

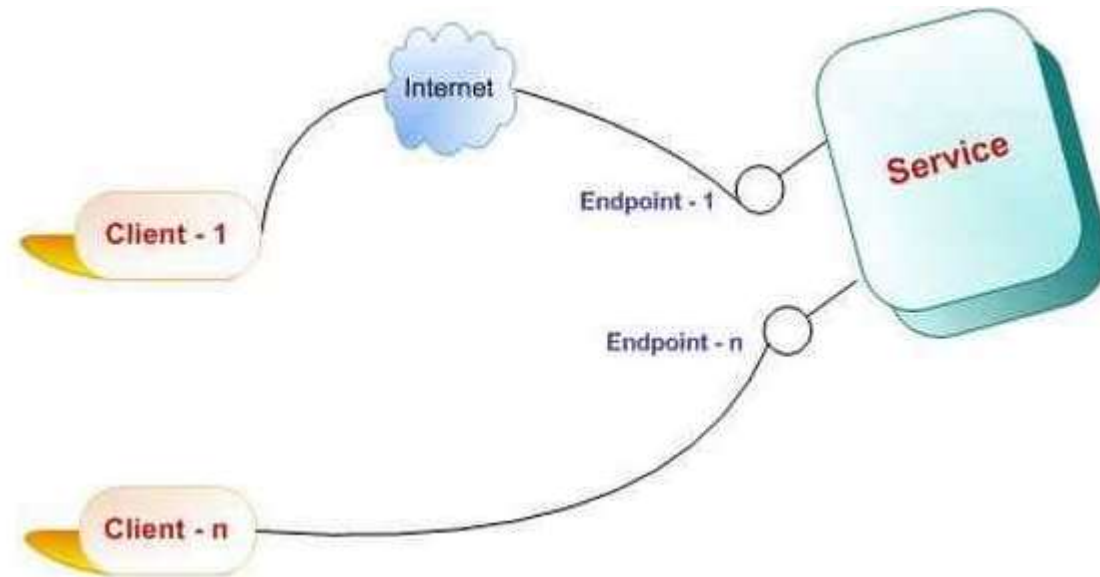
# WCF Communication Protocols

- **HTTP** — Enables you to communicate with WCF services from anywhere, including across the Internet. You can use HTTP communications to create WCF web services.
- **TCP** — Enables you to communicate with WCF services on your local network or across the Internet if you configure your firewall appropriately. TCP is more efficient than HTTP and has more capabilities, but it can be more complicated to configure.
- **UDP** — Similar to TCP in that it enables communications via the local network or Internet, but it's implemented in a subtly different way. One of the consequences of this implementation, which you won't look at in detail, is that a service can broadcast messages to multiple clients simultaneously. The UDP protocol is new the .NET 4.5 Framework, although previously third-party implementations were available.
- **Named pipe** — Enables you to communicate with WCF services that are on the same machine as the calling code, but reside in a separate process.
- **MSMQ** — A queuing technology that enables messages sent by an application to be routed through a queue to arrive at a destination. MSMQ is a reliable messaging technology that ensures that a message sent to a queue will reach that queue. MSMQ is also inherently asynchronous, so a queued message will be processed only when messages ahead of it in the queue have been processed and a processing service is available.



# Endpoints

- It defines the address where a message is to be sent or received. It also specifies the communication mechanism to describe how the messages will be sent along with defining the set of messages. A structure of an endpoint comprises of the following parts
  - Addresses
  - bindings
  - Contracts



# Addresses

- The type of **address** you use for a service depends on the protocol that you are using. Service addresses are formatted for the three protocols described in this slide (MSMQ is not covered) as follows:

➤ **HTTP** — Addresses for the HTTP protocol are URLs of the familiar form `http://<server>:<port>/<service>`.

For SSL connections, you can also use

`https://<server>:<port>/<service>`.

If you are hosting a service in IIS, `<service>` will be a file with a `.svc` extension. IIS addresses will probably include more subdirectories than this example — that is, more sections separated by `/` characters before the `.svc` file.

# Addresses

- **TCP** — Addresses for TCP are of the form `net.tcp://<server>:<port>/<service>`.
- **UDP** — Addresses for UDP are of the form `soap.udp://<server>:<port>/<service>`. Certain `<server>` values are required for multicast communications, but this is beyond the scope of this chapter.
- **Named pipe** — Addresses for named pipe connections are similar but have no port number. They are of the form `net.pipe://<server>/<service>`.

- For example, imagine you create a WCF service with a single operation that has bindings for all three of the protocols listed here. You might use the following base addresses:
  - ✓ `http://www.mydomain.com/services/amazingservices/mygreatservice.svc`
  - ✓ `net.tcp://myhugeserver:8080/mygreatservice`
  - ✓ `net.pipe://localhost/mygreatservice`
- You could then use the following addresses for operations:
  - ✓ `http://www.mydomain.com/services/amazingservices/mygreatservice.svc/greatop`
  - ✓ `net.tcp://myhugeserver:8080/mygreatservice/greatop`
  - ✓ `net.pipe://localhost/mygreatservice/greatop`

# Bindings

- Bindings, specify more than just the transport protocol that will be used by an operation. You can also use them to:
  - specify the security requirements for communication over the transport protocol, . A binding might require username and password authentication or a Windows user account token
  - transactional capabilities of the endpoint,
  - message encoding,
  - and much more.
- Because bindings offer such a great degree of flexibility, the .NET Framework provides some **predefined** bindings that you can use.
- Each binding type is represented by a class in the System.ServiceModel namespace.
- next slide lists these bindings along with some basic information about them.
- In WCF, an endpoint can have multiple *bindings*, each of which specifies a means of communication.

# predefined bindings

BINDING	DESCRIPTION
BasicHttpBinding	The simplest HTTP binding, and the default binding used by web services. It has limited security capabilities and no transactional support.
WSHttpBinding	A more advanced form of HTTP binding that is capable of using all the additional functionality that was introduced in WSE.
WSDualHttpBinding	Extends WSHttpBinding capabilities to include duplex communication capabilities. With duplex communication, the server can initiate communications with the client in addition to ordinary message exchange.
WSFederationHttpBinding	Extends WSHttpBinding capabilities to include federation capabilities. Federation enables third parties to implement single sign-on and other proprietary security measures. This is an advanced topic not covered in this chapter.
NetTcpBinding	Used for TCP communications, and enables you to configure security, transactions, and so on.

# predefined bindings

<code>NetNamedPipeBinding</code>	Used for named pipe communications, and enables you to configure security, transactions, and so on.
<code>NetPeerTcpBinding</code>	Enables broadcast communications to multiple clients, and is another advanced class not covered in this chapter.
<code>NetMsmqBinding</code> and <code>MsmqIntegrationBinding</code>	These bindings are used with MSMQ, which is not covered in this chapter.
<code>NetPeerTcpBinding</code>	Used for peer-to-peer binding, which is not covered in this chapter.
<code>WebHttpBinding</code>	User for web services that use HTTP requests instead of SOAP messages.
<code>NetTcpContextBinding</code>	Similar to <code>NetTcpBinding</code> but allows context information to be exchanged with SOAP headers.
<code>BasicHttpContextBinding</code> and <code>WSHttpContextBinding</code>	Similar to <code>BasicHttpBinding</code> and <code>WSHttpBinding</code> , but allows context information to be exchanged with HTTP cookies or SOAP headers, respectively.
<code>NetHttpBinding</code> and <code>NetHttpsBinding</code>	Similar to <code>BasicHttpBinding</code> and <code>WSHttpBinding</code> , but default to binary message encoding.
<code>UdpBinding</code>	Allows binding to the UDP protocol.



# predefined bindings

- Since .NET 4, endpoints have default bindings that vary according to the protocol used. These defaults are shown in

PROTOCOL	DEFAULT BINDING
HTTP	BasicHttpBinding
TCP	NetTcpBinding
UDP	UdpBinding
Named pipe	NetNamedPipeBinding
MSMQ	NetMsmqBinding



# Contracts

- Contracts define how WCF services can be used. Several types of contract can be defined:
  - **Service contract** — Contains general information about a service and the operations exposed by a service. This includes, for example, the namespace used by service. Services have unique namespaces that are used when defining the schema for SOAP messages in order to avoid possible conflicts with other services.
  - **Operation contract** — Defines how an operation is used. This includes the parameter and return types for an operation method along with additional information, such as whether a method will return a response message.

# Contracts

- **Message contract** — Enables you to customize how information is formatted inside SOAP messages — for example, whether data should be included in the SOAP header or SOAP message body. This can be useful when creating a WCF service that must integrate with legacy systems.
- **Fault contract** — Defines faults that an operation can return. When you use .NET clients, faults result in exceptions that you can catch and deal with in the normal way.
- **Data contract** — If you use complex types, such as user-defined structs and objects, as parameters or return types for operations, then you must define data contracts for these types. Data contracts define the types in terms of the data that they expose through properties.

You typically add contracts to service classes and methods by using attributes

# Example of Endpoints

```
<endpoint address="ServiceUrlGoesHere" binding="TypeOfBinding"  
contract="NameOfContract" />
```

```
<endpoint address=http://localhost:8731/DemoService/  
binding="basicHttpBinding"  
contract=" DemoService.IDemo">
```

# Message Patterns

- An operation contract can define whether an operation returns a value. You've also read about duplex communications that are made possible by the `WSDualHttpBinding` binding. These are both forms of message patterns, of which there are three types:
  - **Request/response messaging** — The “ordinary” way of exchanging messages, whereby every message sent to a service results in a response being sent back to the client. This doesn't necessarily mean that the client waits for a response, as you can call operations asynchronously in the usual way.
  - **One-way, or simplex, messaging** — Messages are sent from the client to the WCF operation, but no response is sent. This is useful when no response is required. For example, you might create a WCF operation that results in the WCF host server rebooting, in which case you wouldn't really want or need to wait for a response.
  - **Two-way, or duplex, messaging** — A more advanced scheme whereby the client effectively acts as a server as well as a client, and the server as a client as well as a server. Once set up, duplex messaging enables both the client and the server to send messages to each other, which might not have responses. This is analogous to creating an object and subscribing to events exposed by that object.

# Behaviors

- Behaviors are a way to apply additional configuration that is not directly exposed to a client to services and operations.
- By adding a behavior to a service, you can control how it is instantiated and used by its hosting process, how it participates in transactions, how multithreading issues are dealt with in the service, and so on.

Operation behaviors can control whether impersonation is used in the operation execution, how the individual operation affects transactions, and more.

Since .NET 4, you can specify **default behaviors** at various levels, so that you don't have to specify every aspect of every behavior for every service and operation. Instead, you can provide defaults and override settings where necessary, which reduces the amount of configuration required. Because this chapter is intended to give you a basic understanding of WCF services, you will only see the most basic functionality of behaviors here.

# Hosting

- WCF services can be hosted in several different processes. These possibilities are as follows:
  - **Web server** — IIS-hosted WCF services are the closest thing to web services that WCF offers. However, you can use advanced functionality and security features in WCF services that are much more difficult to implement in web services. You can also integrate with IIS features such as IIS security.
  - **Executable** — You can host a WCF service in any application type that you can create in .NET, such as console applications, Windows Forms applications, and WPF applications.

# Hosting

- **Windows service** — You can host a WCF service in a Windows service, which means that you can use the useful features that Windows services provide. This includes automatic startup and fault recovery.
- **Windows Activation Service (WAS)** — Designed specifically to host WCF services, WAS is basically a simple version of IIS that you can use where IIS is not available.

# WCF PROGRAMMING

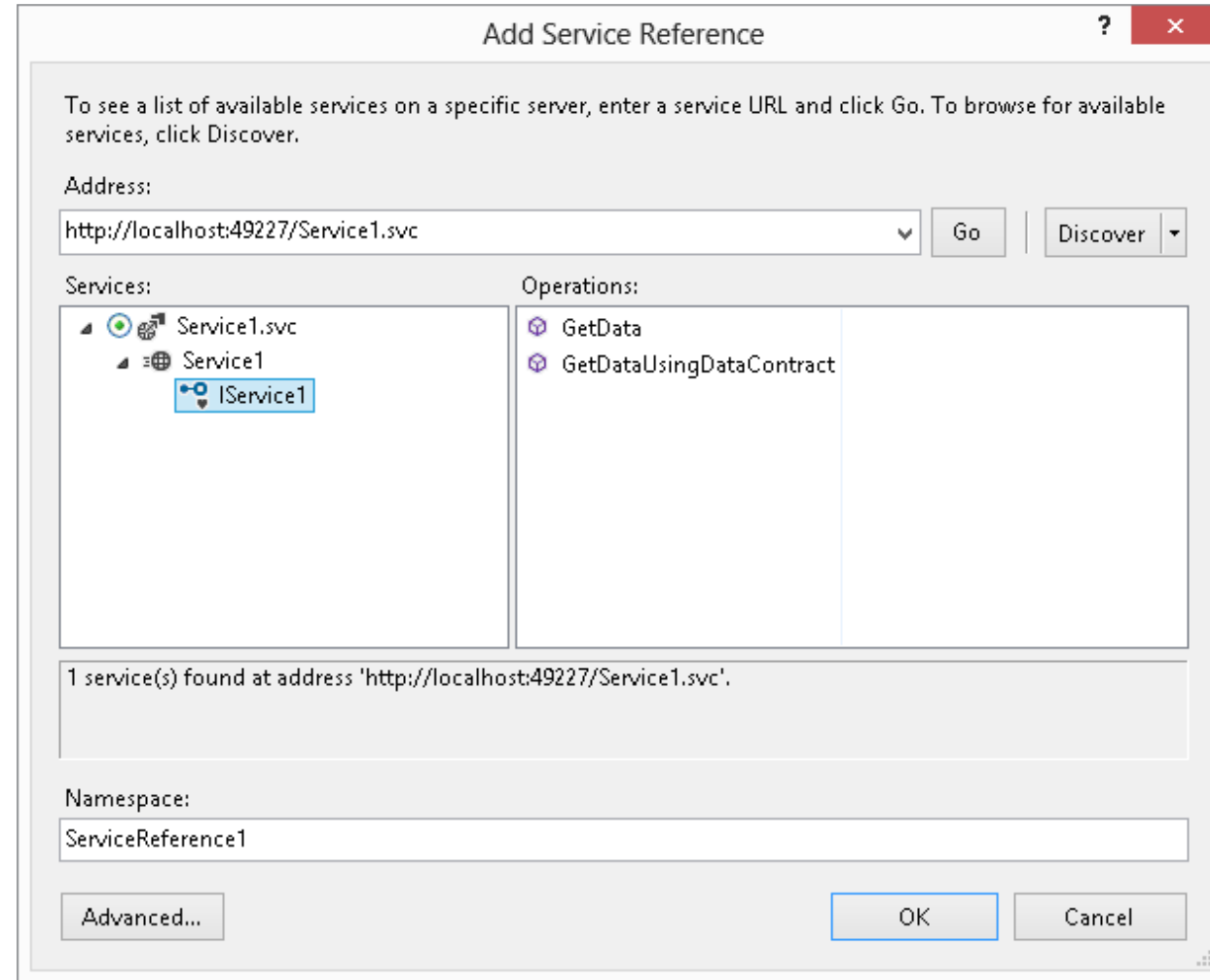
- Now that you have covered all the basics, it is time to get started with some code. In this section you'll start by looking at a simple web server-hosted WCF service and a console application client. After looking at the structure of the code created, you'll learn about the basic structure of WCF services and client applications.



# A Simple WCF Service and Client: Ch25Ex01Client

- 1.** Create a new WCF Service Application project called Ch25Ex01
- 2.** Add a console application called Ch25Ex01Client to the solution.
- 3.** On the Build menu, click Build Solution.
- 4.** Right-click the Ch25Ex01Client project in the Solution Explorer and select Add Service Reference.
- 5.** In the Add Service Reference dialog box, click Discover.

**6.** When the development web server has started and information about the WCF service has been loaded, expand the reference to look at its details, as shown in below Figure (you might have a different port number).



**7.** Click OK to add the service reference.

**8.** Modify the code in Program.cs in the Ch25Ex01Client application as follows:

```
using Ch25Ex01Client.ServiceReference1;
```

```
namespace Ch25Ex01Client
```

```
{    class Program
```

```
{        static void Main(string[] args)
```

```
{            string numericInput = null;
```

```
            int intParam;
```

```
            do
```

```
            {Console.WriteLine("Enter an integer and press enter to call the WCF service.");
```

```
                numericInput = Console.ReadLine();
```

```
            }
```

```
            while (!int.TryParse(numericInput, out intParam));
```

```
            Service1Client client = new Service1Client();
```

```
            Console.WriteLine(client.GetData(intParam));
```

```
            Console.WriteLine("Press an key to exit.");
```

```
            Console.ReadKey();
```

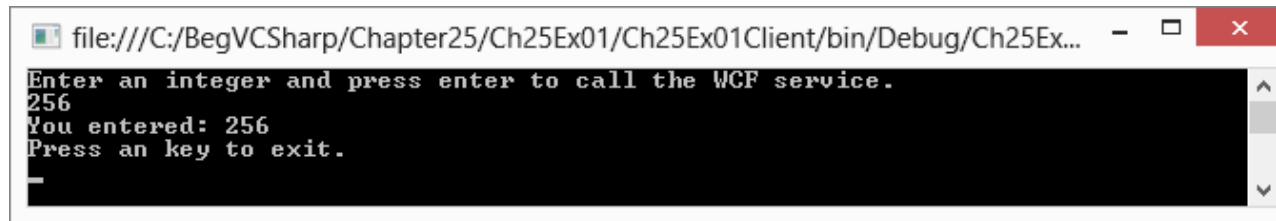
```
        }
```

```
    }
```

```
}
```

**9.** Right-click the Ch25Ex01Client project in the Solution Explorer and select Set as StartUp Project.

**10.** Run the application. Enter a number in the console application window and press Enter. The result is shown in below Figure .

A screenshot of a console application window. The title bar shows the file path: file:///C:/BegVCSharp/Chapter25/Ch25Ex01/Ch25Ex01Client/bin/Debug/Ch25Ex... The console text is as follows:

```
Enter an integer and press enter to call the WCF service.  
256  
You entered: 256  
Press an key to exit.  
_
```

**11.** Exit the application, right-click the Service1.svc file in the Ch25Ex01 project in the Solution Explorer, and click View in Browser.

**12.** Review the information in the window

**13.** Click the link at the top of the web page for the service to view the WSDL. Don't panic — you don't need to understand all the stuff in the WSDL file!

# The WCF Test Client

- the client application you want to use might be complex, and it can be tricky to test services properly.
- To ease the development of WCF services, VS provides a **test tool** you can use to ensure that your WCF operations work correctly. This tool is automatically configured to work with your WCF service projects, so if you run your project the tool will appear. Alternatively, you can run the test client as a standalone application.
- You can find the test client on 64-bit operating systems at:  
C:\ProgramFiles(x86)\MicrosoftVisualStudio11.0\Common7\IDE\WcfTestClient.exe.

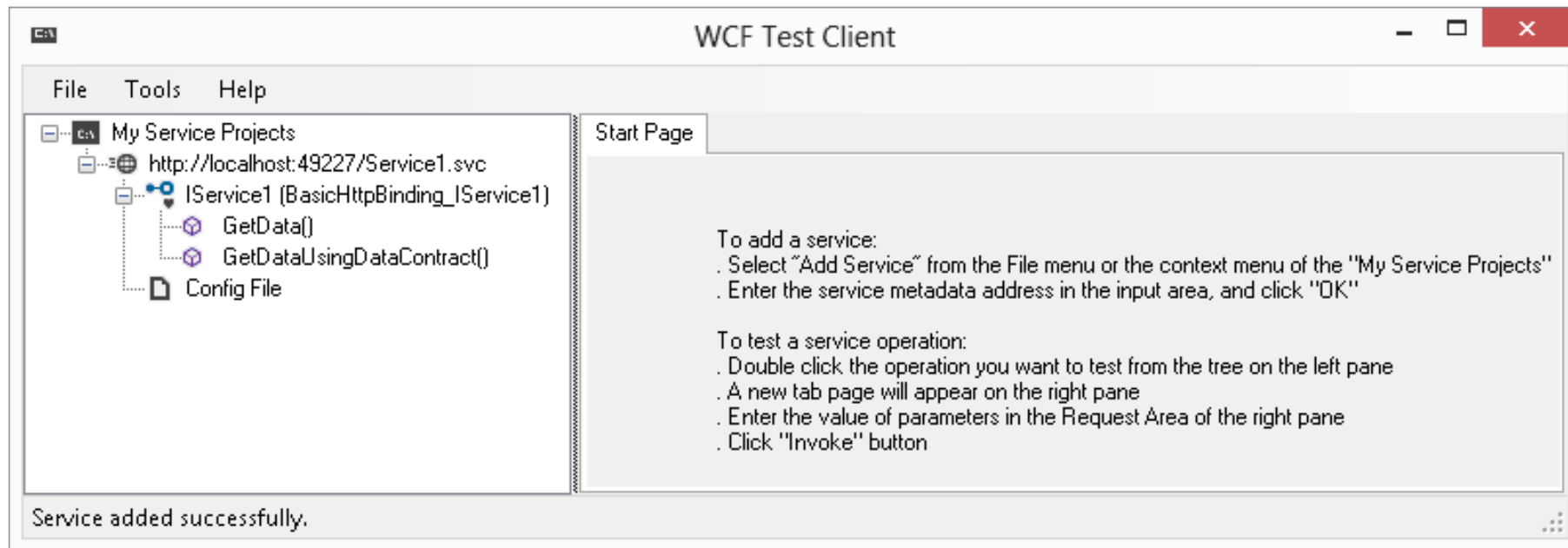
# Using the WCF Test Client:Ch25Ex01\Web.config

- **1.** Open the WCF Service Application project from the previous Try It Out, Ch25Ex01.
- **2.** Right-click the Service1.svc service in Solution Explorer and click Set As Start Page.
- **3.** Right-click the Ch25Ex01 project in Solution Explorer and click Set As StartUp Project.
- **4.** In Web.config, ensure that metadata is enabled:

```
<serviceMetadata httpGetEnabled="true" httpsGetEnabled="true"  
/>
```

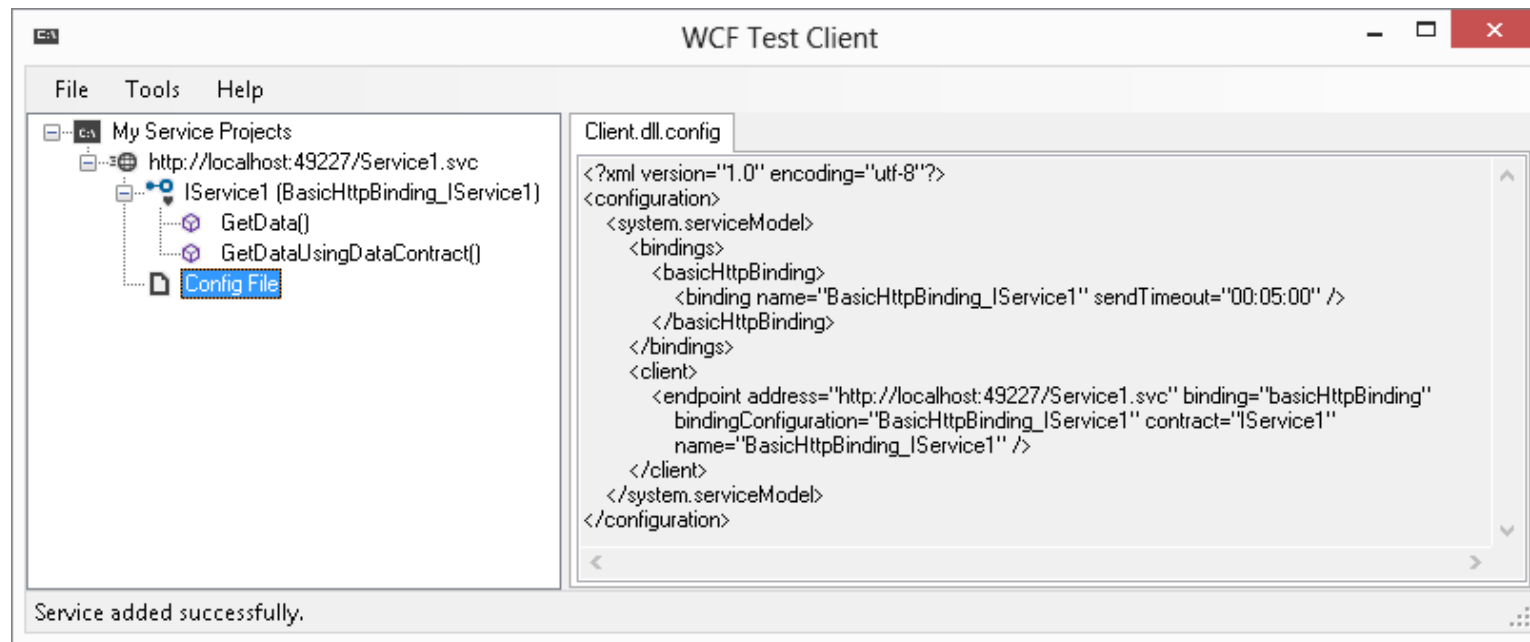
# Using the WCF Test Client:Ch25Ex01\Web.config

5. Run the application. The WCF test client appears, as shown in Figure(it takes a moment or two to add the service).



# Using the WCF Test Client:Ch25Ex01\Web.config

6. In the left pane of the test client, double-click Config File. The config file used to access the service is displayed in the right pane



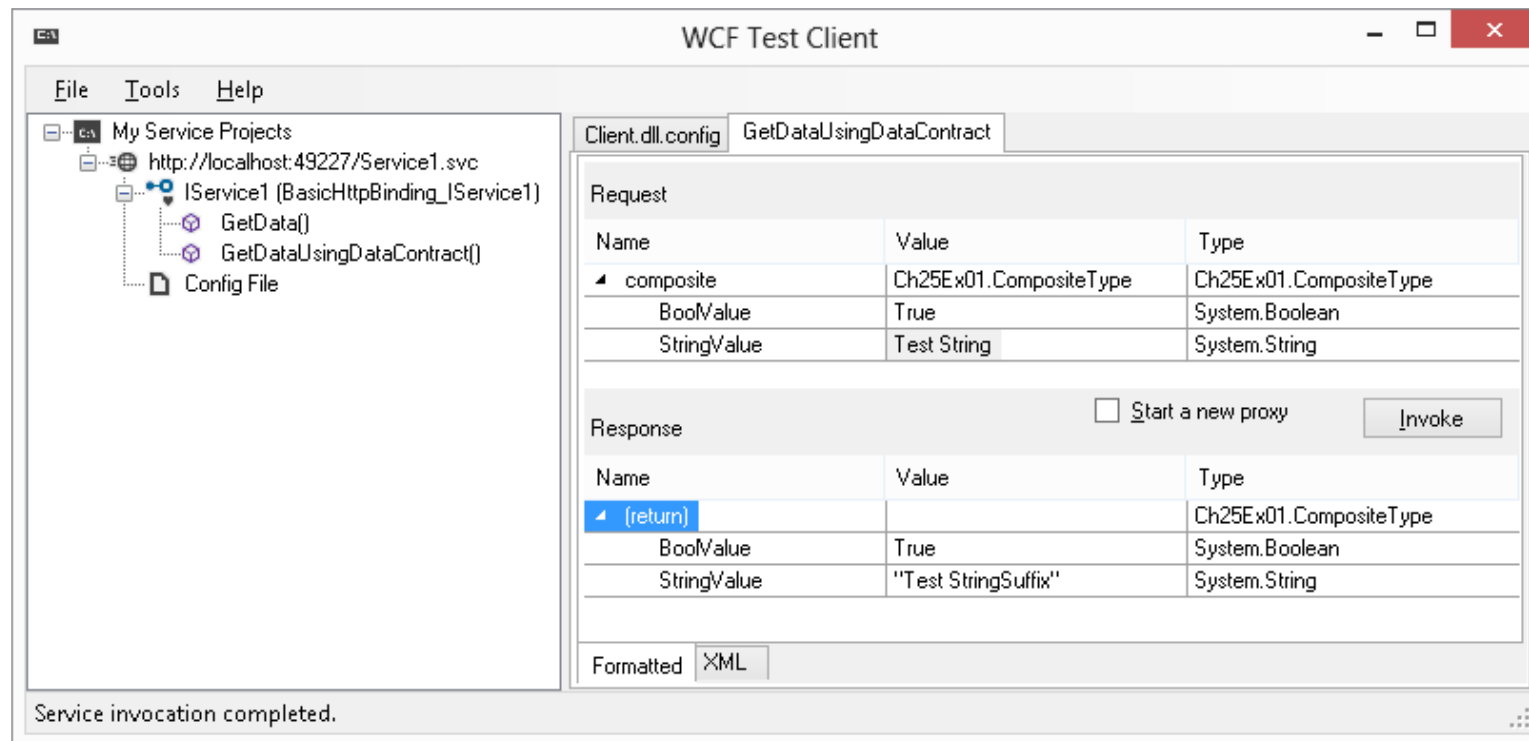


# Using the WCF Test Client:Ch25Ex01\Web.config

- 7.** In the left pane, double-click the GetDataUsingDataContract() operation.
- 8.** In the pane that appears on the right, change the value of BoolValue to True and StringValue to Test String, and then click Invoke.
- 9.** If a security prompt dialog box appears, click OK to confirm that you are happy to send information to the service.

# Using the WCF Test Client:Ch25Ex01\Web.config

**10.** The operation result appears, as shown in Figure 25-7.



# Using the WCF Test Client:Ch25Ex01\Web.config

**11.** Click the XML tab to view the request and response XML, shown in Figure

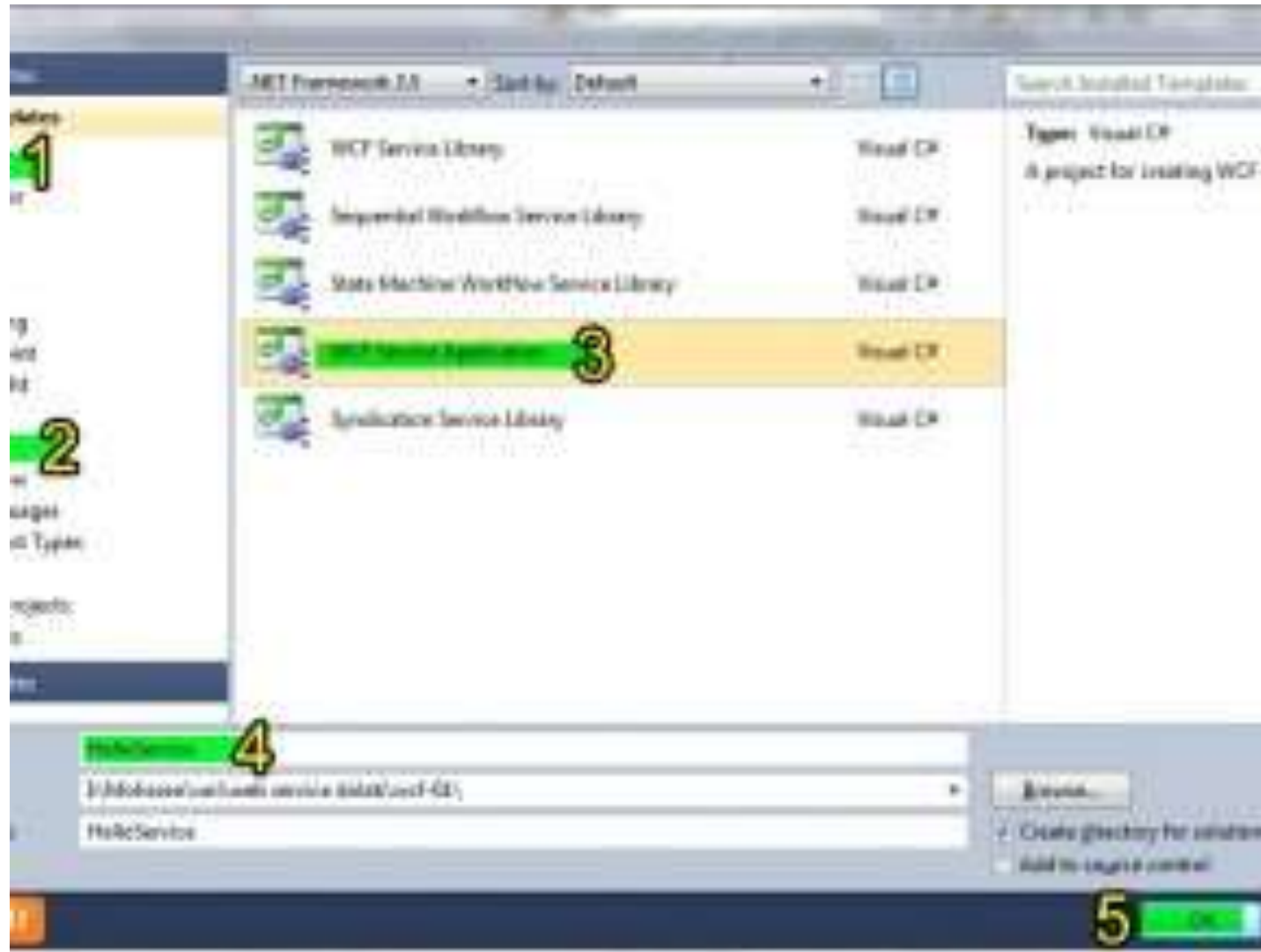


# Defining WCF Service Contracts

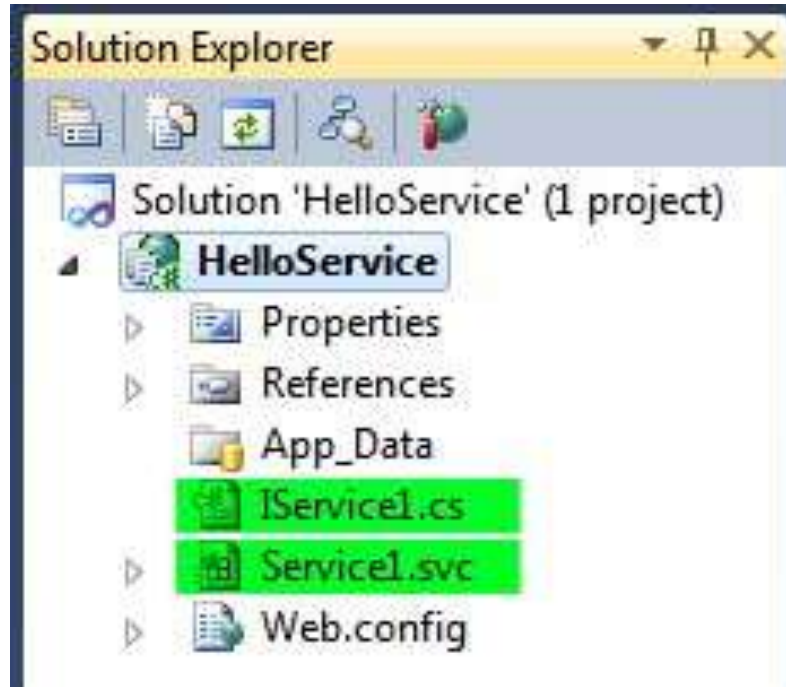
- The previous examples showed how the WCF infrastructure makes it easy for you to define contracts for WCF services with a combination of classes, interfaces, and attributes. This section takes a deeper look at this technique.

• WCF Service ایجاد کنیم که نام و زبان خود را به آن ارسال کنیم و به زبان ارسالی به ما سلام کند.

- نخست ویژوال استودیو را اجرا کرده و پس از آن از منوی File و زیرمنوی New، روی Project کلیک کنید. از پنجره‌ی باز شده و از سمت چپ Visual C# و زیر دسته‌ی WCF را انتخاب کرده و سپس از قسمت میانی پنجره WCF service application را انتخاب کرده و نامی برای سرویس خود برگزینید که در اینجا من نام HelloService را بکار بردم.

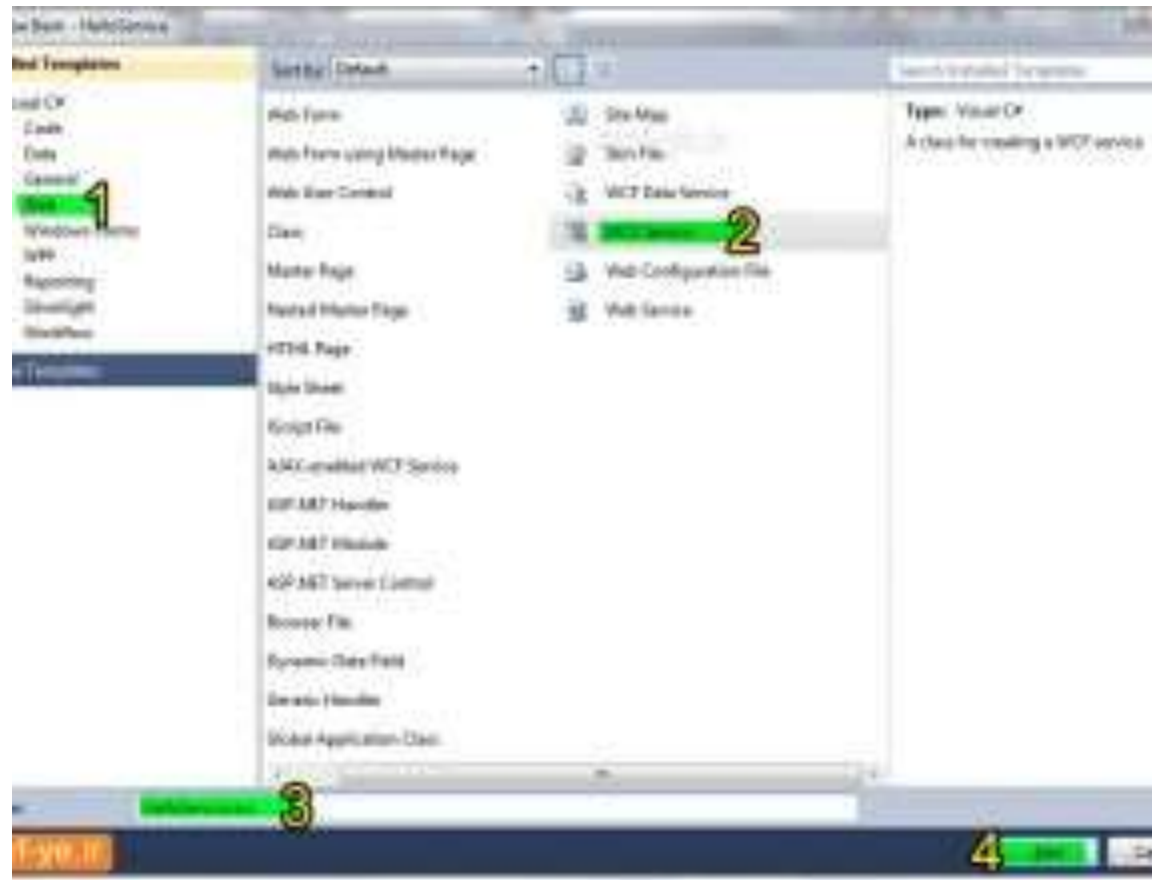


- ویژوال استودیو همه‌ی فایل‌های لازم را برای شروع کار می‌سازد. بصورت پیش‌فرض سرویس نام `Service1.svc` و رابط `IService.cs` را دارد. شکل زیر نمایی از پنجره `Solution Explorer` پروژه‌ی ایجاد شده است:



- در اینجا به این دو پیش‌فرض نیاز نداریم، یا باید حذف کنیم و یا برای ادامه‌ی کار، این دو را ویرایش کنیم. من در اینجا هر دو را با کلیک راست کردن روی هر کدام، حذف (Delete) می‌کنم.

- حال این پروژه برای سرویس‌دهی آماده نیست، پس برای ساختن سرویس از پنجره Solution Explorer روی نام پروژه (HelloService) کلیک راست کرده، Add را انتخاب و سپس روی New Item کلیک می‌کنیم.
- در پنجره‌ی باز شده WCF service را انتخاب کرده و نام HelloService.svc را وارد کرده و روی دکمه‌ی Add کلیک می‌کنیم.



- حالا ویژوال استودیو دو فایل با نام‌های `HelloService.cs` و `HelloService.svc` ساخته است. این فایل‌ها در واقع برای تعریف سرویس (`IHelloService.cs`) و اجرای سرویس (`HelloService.svc`) هستند.
- اجازه دهید که در اینجا رابط `IHelloService.cs` را بررسی کنیم، شکل زیر مربوط به محتویات `IHelloService` است:

```
namespace HelloService
{
    // NOTE: You can use the "Rename"
    [ServiceContract]
    public interface IHelloService
    {
        [OperationContract]
        void DoWork();
    }
}
```

- کدهای موجود در `IHelloService` برای تعریف سرویس و فراهم‌کننده‌های اپریشن/متد با نام `DoWork()` است. هر سرویس `WCF`، فایل رابطی را دارد که سرویس را تعریف می‌کند.



- و حالا اجازه دهید که فایل `HelloService.svc` را بررسی کنیم که شکل زیر مربوط به آن است:

```
namespace HelloService

{
    // NOTE: You can use the "Rename" command
    public class HelloService : IHelloService
    {
        public void DoWork()
        {
        }
    }
}
```

- کد نشان داده شده برای اجرای رابط `HelloService` است. کلاس `HelloService` نیاز به اجرای همه‌ی متدهای مشخص شده در تعریفات سرویس را دارد.

- **IServiceContract** را در محیط ویژوال استودیو با کلیک کردن بر روی نام آن باز کنید و متد **DoWork()** را حذف کرده و یک تعریف برای **SayHello()** ایجاد کنید. اضافه کردن **OperationContract** به متد را فراموش نکنید. گدهای لازم برای این کار به شکل زیر می باشد:

```
using System.Runtime.Serialization;  
using System.ServiceModel;  
using System.Text;
```

```
namespace HelloService  
{  
    [ServiceContract]  
    public interface IHelloService  
    {  
        [OperationContract]  
        string SayHello(String name, String language);  
    }  
}
```

- حالا برای پیاده‌سازی متد SayHello فایل HelloService.svc.cs را ویرایش کرده و کدهای زیر بنویسید:

```
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
namespace HelloService
{
    public class HelloService : IHelloService
    {
        public string SayHello(String name, String language)
        {
            switch (language)
            {
                case "en":
                    return "Hello " + name;
                case "fa":
                    return سلام +name;
                default:
                    return "زبان پشتیبانی نشده"
            }
        }
    }
}
```

- ما موفق به ساخت این سرویس شده‌ایم.
- حالا برای اجرای این سرویس می‌توان کلید F5 را از صفحه‌کلید فشار داده و یا از منوی Debug روی گزینه‌ی start debugging کلیک کنید.
- اگر همه‌ی مراحل بالا را به درستی انجام داده باشید؛ ویژوال استودیو میزبانی سرویس شما را به WCF Test Client می‌دهد که قبلاً با آن آشنا شده‌اید و شکل زیر نمایش داده می‌شود:

