http://www.codeproject.com/Articles/97204/Implementing-a-Basic-Hello-World-WCF-Service

Implementing a Basic Hello World WCF Service

By Mike_Liu, 18 May 2013

Download source code - 42.5 KB

Update

An updated version of this article for .NET 4.5 can be found at ***Implementing a Basic Hello World WCF Service (v4.5)***

Introduction

This is the first article I've written on WCF/LINQ. In the first three articles on CodeProject.com, I'll explain the fundamentals of Windows Communication Foundation (WCF), including:

- Implementing a Basic Hello World WCF Service (this article)
- Implementing a WCF Service with Entity Framework
- Concurrency Control of a WCF Service with Entity Framework

In the other articles, I'll explain LINQ, LINQ to SQL, Entity Framework, and LINQ to Entities. Followings are the articles I wrote for LINQ, LINQ to SQL, and LINQ to Entities:

- Introducing LINQ—Language Integrated Query
- LINQ to SQL: Basic Concepts and Features
- LINQ to SQL: Advanced Concepts and Features
- LINQ to Entities: Basic Concepts and Features

Overview

In this article, we will manually implement a basic WCF Service from scratch, step by step, with clear instructions and precise screen snapshots. You will have a thorough understanding of what WCF is under the hood after you read this article. Visual Studio 2010 under Windows 7 will be used for all screenshots of this article.

We will build the WCF Service manually from scratch, meaning we will not use any Visual Studio 2010 template to create the service. We will also create the host application and the test client application manually, including generating the proxy and configuration files manually with the tool *svcutils.exe*. In your real project, you can and should utilize Visual Studio 2010 to help with these tasks, but manually doing the actual work is a great way for you to understand what WCF is really like under the hood. This will help you to better understand the why of those WCF templates within Visual Studio.

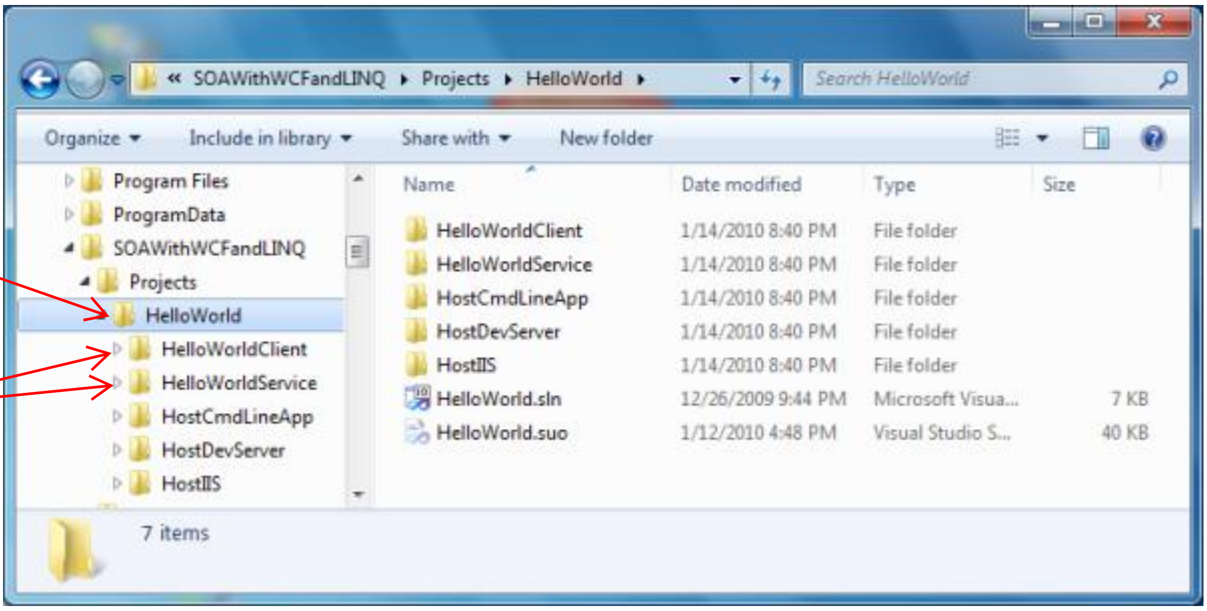
We will build a HelloWorld WCF Service by carrying out the following steps:

- Create the solution and project
- Create the WCF service contract interface
- Implement the WCF Service
- Host the WCF Service in the ASP.NET Development Server
- Create a client application to consume this WCF Service

Creating the HelloWorld solution and project

Before we can build the WCF Service, we need to create a solution for our service projects. We also need a directory in which to save all the files. Throughout this article, we will save our project source code in the*C:\SOAwithWCFandLINQ\Projects* directory. We will have a subfolder for each solution we create, and under this solution folder, we will have a subfolder for each project.

For this HelloWorld solution, the final directory structure is shown in the following image:

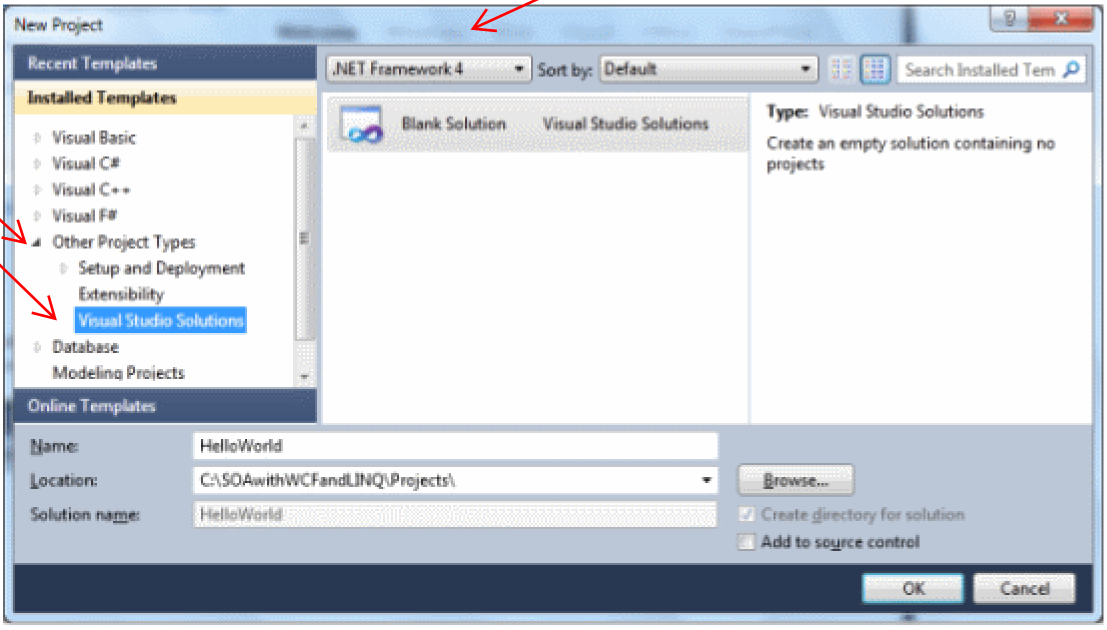


Name	Date modified	Type	Size
HelloWorldClient	1/14/2010 8:40 PM	File folder	
HelloWorldService	1/14/2010 8:40 PM	File folder	
HostCmdLineApp	1/14/2010 8:40 PM	File folder	
HostDevServer	1/14/2010 8:40 PM	File folder	
HostIIS	1/14/2010 8:40 PM	File folder	
HelloWorld.sln	12/26/2009 9:44 PM	Microsoft Visua...	7 KB
HelloWorld.suo	1/12/2010 4:48 PM	Visual Studio S...	40 KB

Note: you don't need to manually create these directories via Windows Explorer; Visual Studio will create them automatically when you create the solutions and projects.

Now, follow these steps to create our first solution and the HelloWorld project:

1. Start Visual Studio 2010. If the Open Project dialog box pops up, click Cancel to close it.
2. Go to menu File | New | Project. The New Project dialog window will appear.



Type: Visual Studio Solutions

Create an empty solution containing no projects

Name: HelloWorld

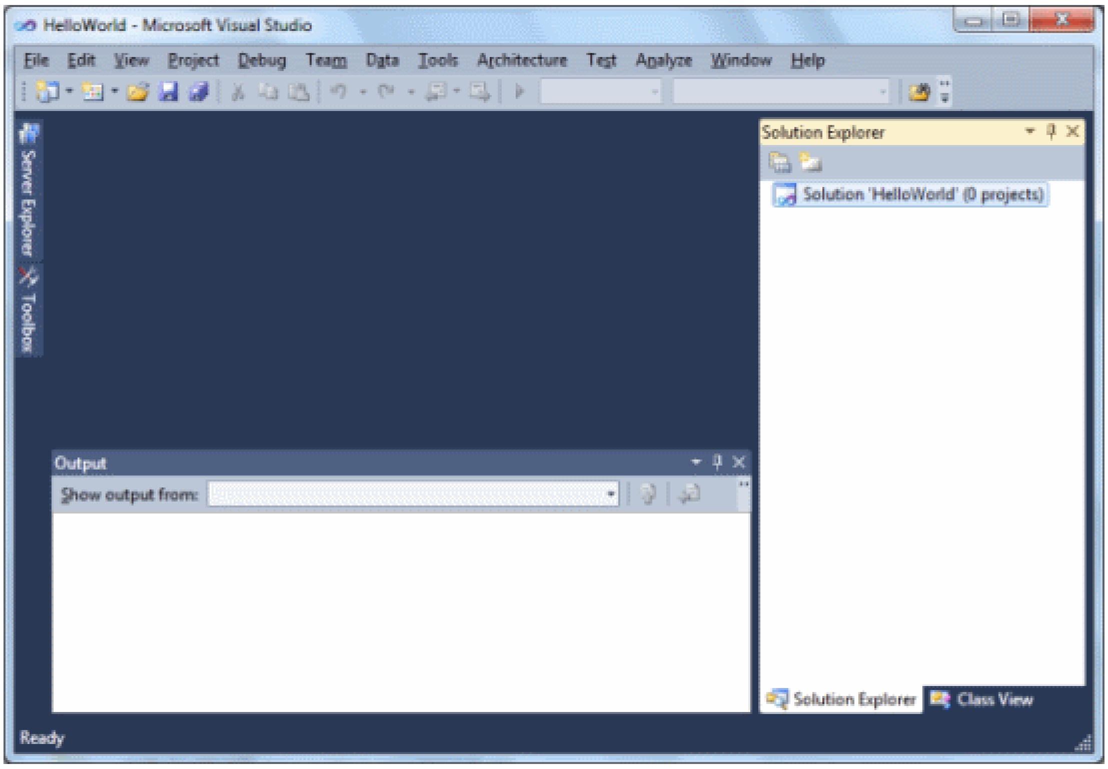
Location: C:\SOAwithWCFandLINQ\Projects\

Solution name: HelloWorld

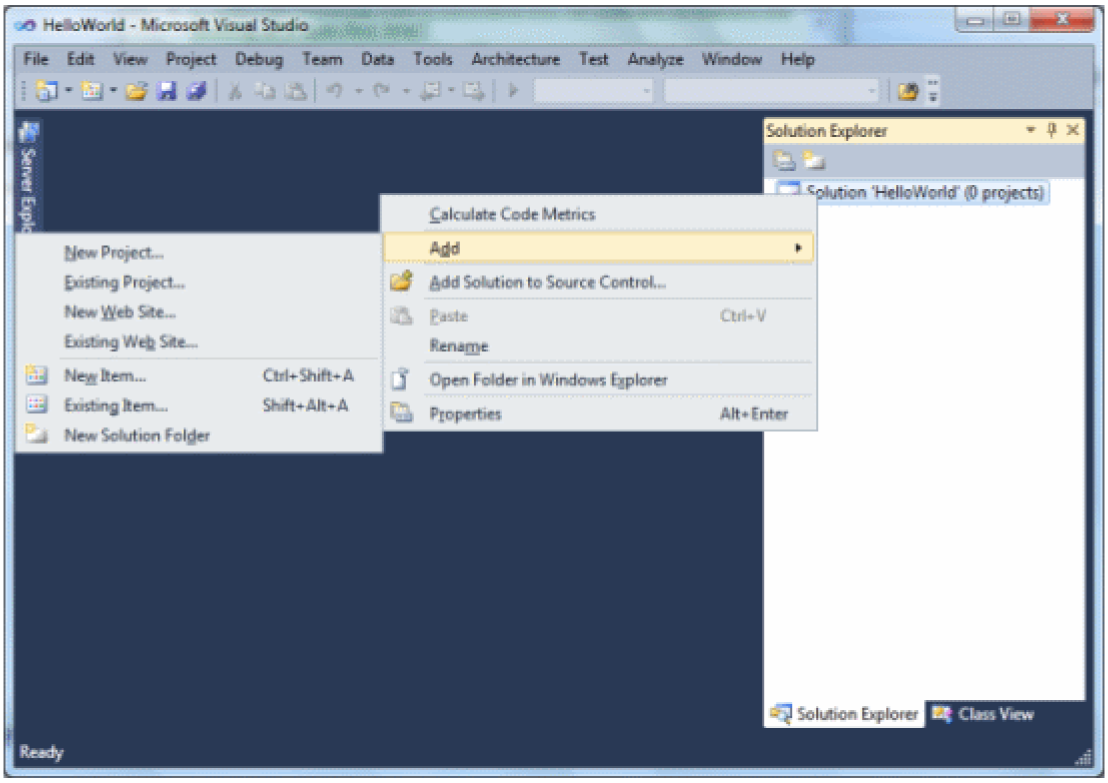
☑ Create directory for solution

☐ Add to source control

3. From the left-hand side of the window (Installed Templates), expand Other Project Types and then select Visual Studio Solutions as the template. From the middle side of the window, select Blank Solution.
4. At the bottom of the window, type HelloWorld as the Name, and *C:\SOAwithWCFandLINQ\Projects\* as the Location. Note that you should not enter HelloWorld within the location, because Visual Studio will automatically create a folder for a new solution.
5. Click the OK button to close this window, and your screen should look like the following image, with an empty solution.

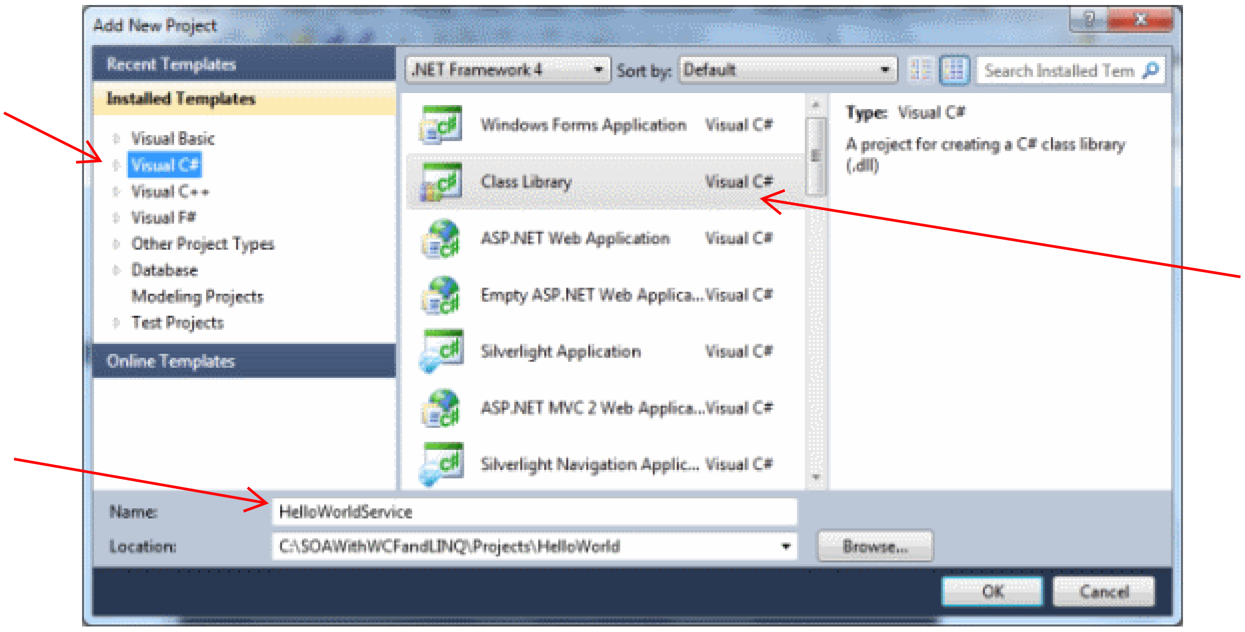

HelloWorld - Microsoft Visual Studio
File Edit View Project Debug Team Data Tools Architecture Test Analyze Window Help
Server Explorer
Toolbox
Solution Explorer
Solution 'HelloWorld' (0 projects)
Output
Show output from:
Solution Explorer
Class View
Ready

6. Depending on your settings, the layout may be different. But you should still have an empty solution in your Solution Explorer. If you don't see Solution Explorer, go to menu View | Solution Explorer, or press Ctrl+Alt+L to bring it up.
7. In the Solution Explorer, right-click on the solution, and select Add | New Project... from the context menu. You can also go to menu File | Add | New Project... to get the same result. The following image shows the context menu for adding a new project:


HelloWorld - Microsoft Visual Studio
File Edit View Project Debug Team Data Tools Architecture Test Analyze Window Help
Server Explorer
Solution Explorer
Solution 'HelloWorld' (0 projects)
Calculate Code Metrics
Add
Add Solution to Source Control...
Paste Ctrl+V
Rename
Open Folder in Windows Explorer
Properties Alt+Enter
New Project...
Existing Project...
New Web Site...
Existing Web Site...
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Solution Folder
Solution Explorer
Class View
Ready

8. The Add New Project window should now appear on your screen. In the left-hand side of this window (Installed Templates), select Visual C# as the template, and on the middle side of the window, select Class Library.

9. At the bottom of the window, type HelloWorldService as the Name. LeaveC:*SOAwithWCFandLINQ**Projects**HelloWorld* as the Location. Again, don't add HelloWorldService to the location, as Visual Studio will create a subfolder for this new project (Visual Studio will use the solution folder as the default base folder for all the new projects added to the solution).



Name: HelloWorldService
Location: C:\SOAWithWCFandLINQ\Projects\HelloWorld

You may have noticed that there is already a template for WCF Service Application in Visual Studio 2010. For this very first example, we will not use this template. Instead, we will create everything by ourselves so you know what the purpose of each template is. This is an excellent way for you to understand and master this new technology.

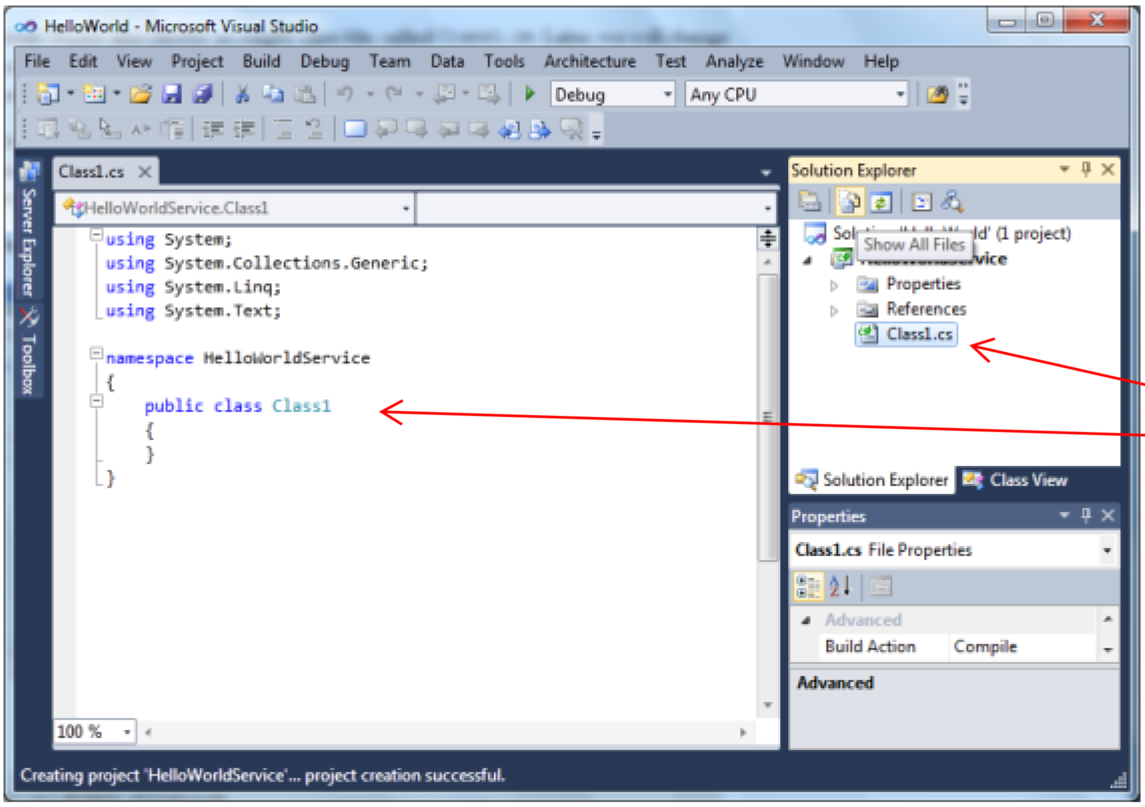
10. Now, you can click the OK button to close this window.

Once you click the OK button, Visual Studio will create several files for you. The first file is the project file. This is an XML file under the project directory, and it is called *HelloWorldService.csproj*.

Visual Studio also creates an empty class file, called *Class1.cs*. Later, we will change this default name to a more meaningful one, and change its namespace to our own.

Three directories are created automatically under the project folder: one to hold the binary files, another to hold the object files, and a third one for the property files of the project.

The window on your screen should now look like the following image:



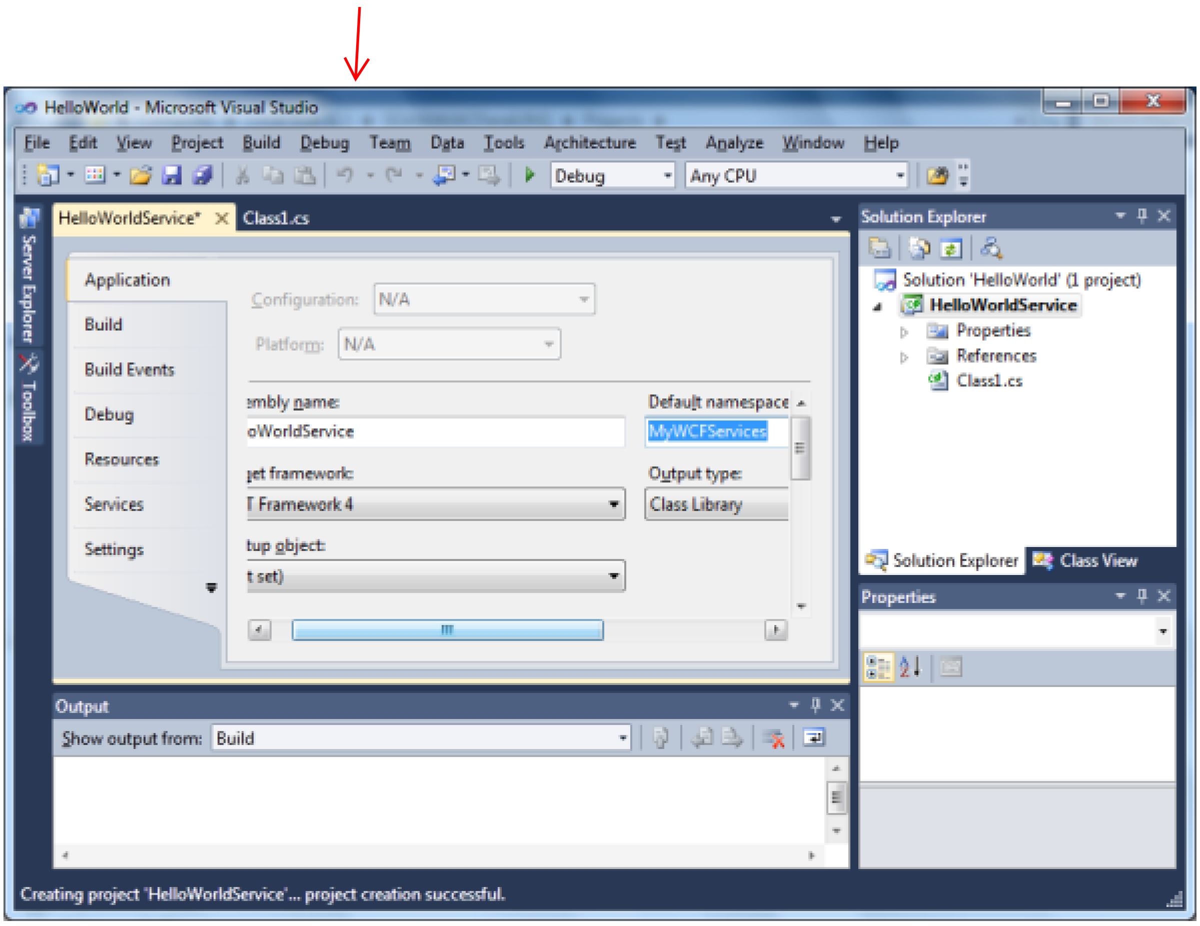
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace HelloWorldService
{
    public class Class1
    {
    }
}
```

Creating project 'HelloWorldService'... project creation successful.

We now have a new solution and project created. Next, we will develop and build this service. But before we go any further, we need to do two things to this project:

1. Click the Show All Files button on the Solution Explorer toolbar. It is the second button from the left, just above the word Solution inside the Solution Explorer. If you allow your mouse to hover above this button, you will see the hint Show All Files, as shown in the above diagram. Clicking this button will show all the files and directories in your hard disk under the project folder - even those items that are not included in the project. Make sure that you don't have the solution item selected. Otherwise, you can't see the Show All Files button.
2. Change the default namespace of the project. From the Solution Explorer, right-click on the HelloWorldService project, select Properties from the context menu, or go to menu item Project | HelloWorldService Properties... You will see the project properties dialog window. On the Application tab, change Default namespace to `MyWCFServices`.



Configuration: N/A

Platform: N/A

…mbly name: …oWorldService

Default namespace: MyWCFServices

…get framework: …T Framework 4

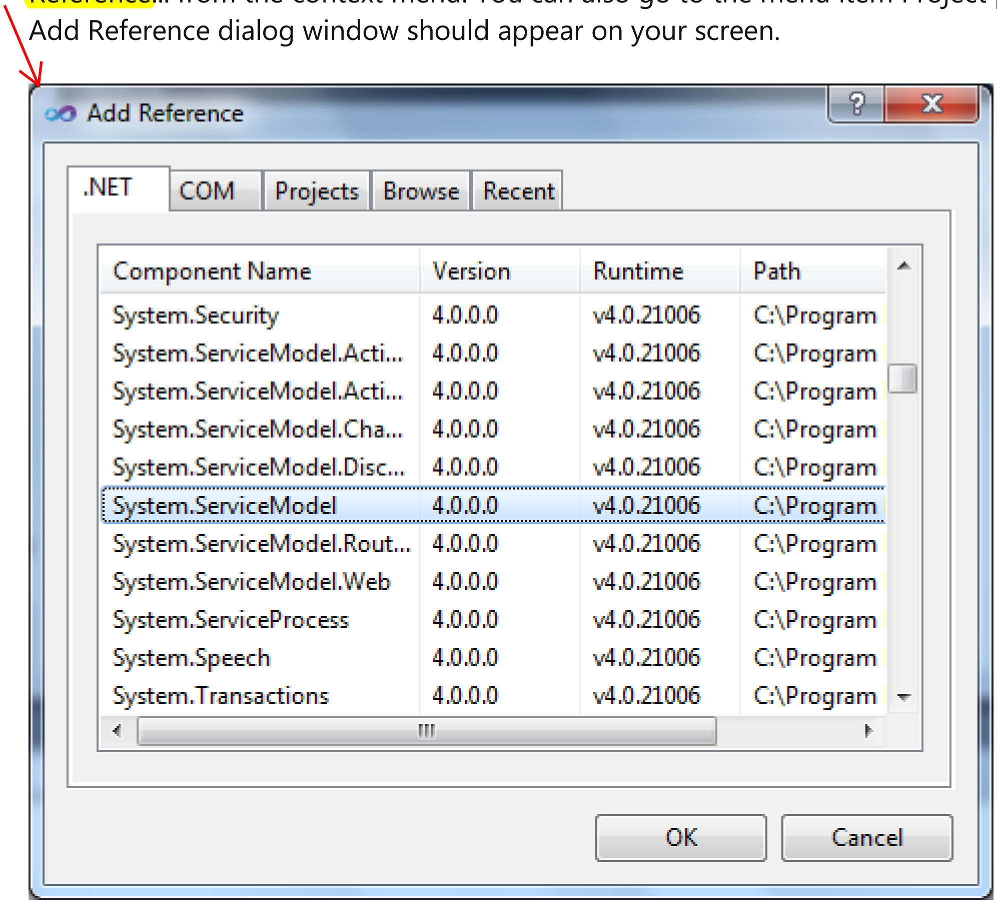
Output type: Class Library

…tup object: …t set)

Creating project 'HelloWorldService'... project creation successful.

Lastly, in order to develop a WCF Service, we need to add a reference to the `System.ServiceModel` namespace.

1. On the Solution Explorer window, right-click on the HelloWorldService project, and select Add Reference... from the context menu. You can also go to the menu item Project | Add Reference... to do this. The Add Reference dialog window should appear on your screen.



Component Name	Version	Runtime	Path
System.Security	4.0.0.0	v4.0.21006	C:\Program
System.ServiceModel.Acti...	4.0.0.0	v4.0.21006	C:\Program
System.ServiceModel.Acti...	4.0.0.0	v4.0.21006	C:\Program
System.ServiceModel.Cha...	4.0.0.0	v4.0.21006	C:\Program
System.ServiceModel.Disc...	4.0.0.0	v4.0.21006	C:\Program
System.ServiceModel	4.0.0.0	v4.0.21006	C:\Program
System.ServiceModel.Rout...	4.0.0.0	v4.0.21006	C:\Program
System.ServiceModel.Web	4.0.0.0	v4.0.21006	C:\Program
System.ServiceProcess	4.0.0.0	v4.0.21006	C:\Program
System.Speech	4.0.0.0	v4.0.21006	C:\Program
System.Transactions	4.0.0.0	v4.0.21006	C:\Program

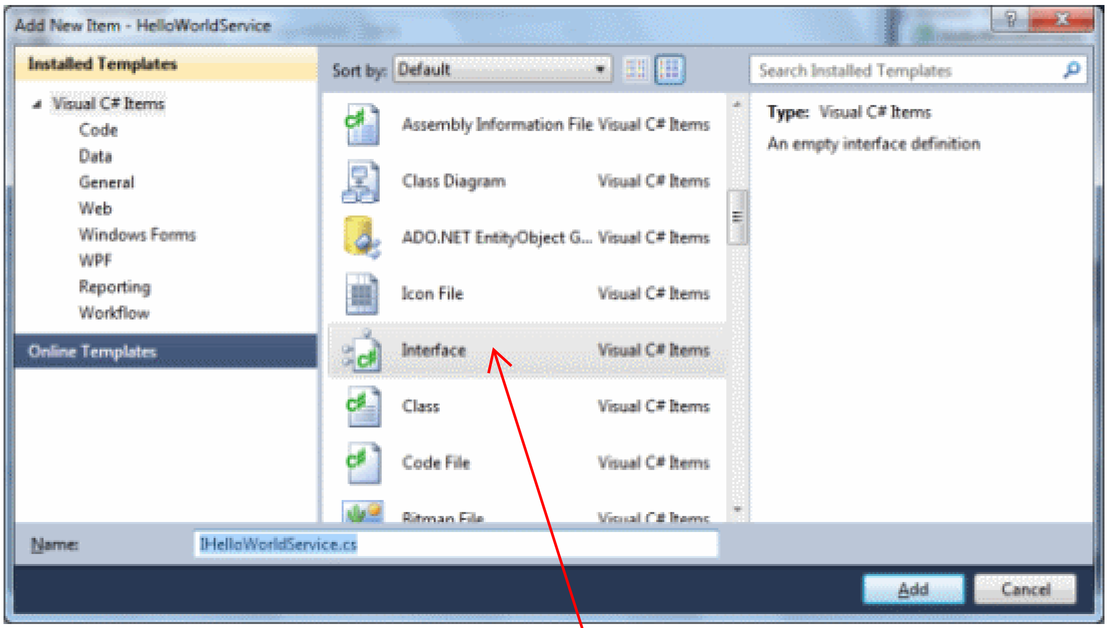
2. Select System.ServiceModel from the .NET tab, and click OK.

Now, on the Solution Explorer, if you expand the references of the HelloWorldService project, you will see that`System.ServiceModel` has been added. Also note that `System.Xml.Linq` is added by default. We will use this later when we query a database.

Creating the HelloWorldService service contract interface

In the previous section, we created the solution and the project for the HelloWorld WCF Service. From this section on, we will start building the HelloWorld WCF Service. First, we need to create the service contract interface.

1. In the Solution Explorer, right-click on the HelloWorldService project, and select Add | New Item... from the context menu. The following Add New Item - HelloWorldService dialog window should appear on your screen:


Add New Item - HelloWorldService
Installed Templates
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
Reporting
Workflow
Online Templates
Sort by: Default
Search Installed Templates
Assembly Information File Visual C# Items
Class Diagram Visual C# Items
ADO.NET EntityObject G... Visual C# Items
Icon File Visual C# Items
Interface Visual C# Items
Class Visual C# Items
Code File Visual C# Items
Bitmap File Visual C# Items
Type: Visual C# Items
An empty interface definition
Name: IHelloWorldService.cs
Add
Cancel

2. On the left-hand side of the window (Installed Templates), select Visual C# Items as the template, and on the middle side of the window, select Interface.
3. At the bottom of the window, change the Name from *Interface1.cs* to *IHelloWorldService.cs*.
4. Click the Add button.

Now, an empty service interface file has been added to the project. Follow the steps below to customize it:

1. Add a `using` statement:

⊟ Collapse | Copy Code

```
using System.ServiceModel;
```

2. Add a `ServiceContract` attribute to the interface. This will designate the interface as a WCF service contract interface.

⊟ Collapse | Copy Code

```
[ServiceContract]
```

3. Add a `GetMessage` method to the interface. This method will take a string as the input, and return another string as the result. It also has an attribute, `OperationContract`.

⊟ Collapse | Copy Code

```
[OperationContract]
String GetMessage(String name);
```

4. Change the interface to `public`.

The final content of the file *IHelloWorldService.cs* should look like the following:

⊟ Collapse | Copy Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel;
namespace MyWCFServices
{
    [ServiceContract]
    public interface IHelloWorldService
    {
        [OperationContract]
        String GetMessage(String name);
    }
}
```

Implementing the HelloWorldService service contract

Now that we have defined a service contract interface, we need to implement it. For this purpose, we will re-use the empty class file that Visual Studio created for us earlier, and modify this to make it the implementation class of our service.

Before we modify this file, we need to rename it. In the Solution Explorer window, right-click on the file *Class1.cs*, select Rename from the context menu, and rename it to *HelloWorldService.cs*.

Note: Visual Studio is smart enough to change all the related files and references to use this new name. You can also select the file and change its name from the Properties window.

Next, follow the steps below to customize this class file.

1. Change its namespace from `HelloWorldService` to `MyWCFServices`. This is because this file was added before we changed the default namespace of the project.
2. Make it inherit from the interface `IHelloWorldService`.

⊟ Collapse | Copy Code

```
public class HelloWorldService: IHelloWorldService
```

3. Add a `GetMessage` method to the class. This is an ordinary C# method that returns a string.

⊟ Collapse | Copy Code

```
public String GetMessage(String name)
{
    return "Hello world from " + name + "!";
}
```

The final content of the file *HelloWorldService.cs* should look like the following:

⊟ Collapse | Copy Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace MyWCFServices
{
    public class HelloWorldService: IHelloWorldService
    {
        public String GetMessage(String name)
        {
            return "Hello world from " + name + "!";
        }
    }
}
```

Now, build the project. If there is no build error, it means that you have successfully created your first WCF Service. If you see a compilation error, such as "'ServiceModel' does not exist in the namespace 'System'", this is probably because you didn't add the `System.ServiceModel` namespace reference correctly. Revisit the previous section to add this reference, and you are all set.

Next, we will host this WCF Service in an environment and create a client application to consume it.

Hosting the WCF Service in an ASP.NET Development Server

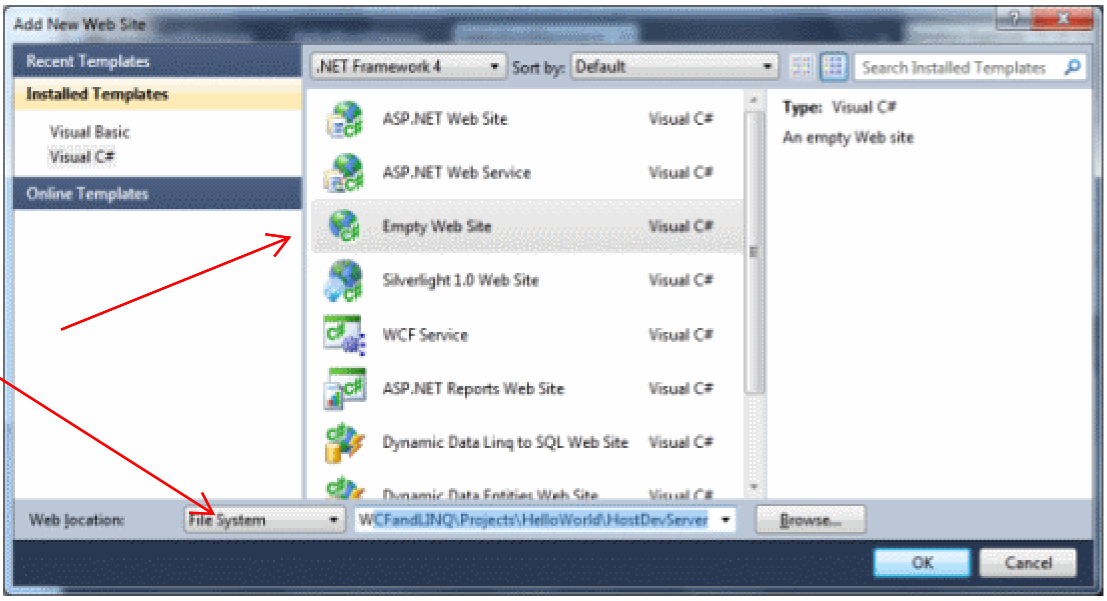
The HelloWorldService is a class library. It has to be hosted in an environment so that client applications may access it. In this section, we will explain how to host it using the ASP.NET Development Server. Later, in the next section, we will discuss more hosting options for a WCF Service.

Creating the host application

There are several built-in host applications for WCF Services within Visual Studio 2010. However, in this section, we will manually create the host application so that you can have a better understanding of what a hosting application is really like under the hood. In subsequent chapters, we will explain and use the built-in hosting application.

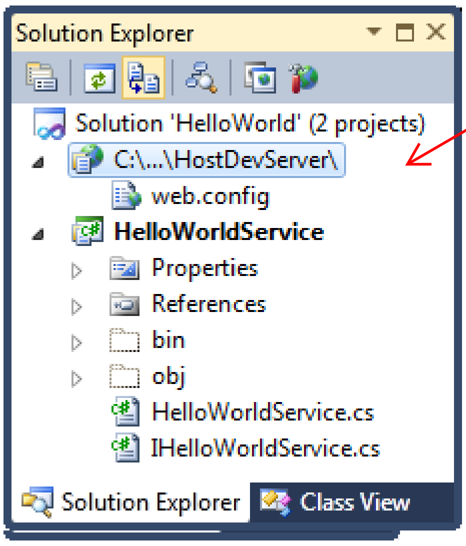
To host the library using the ASP.NET Development Server, we need to add a new web site to the solution. Follow these steps to create this web site:

1. In the Solution Explorer, right-click on the solution file, and select Add | New Web Site... from the context menu. The Add New Web Site dialog window should pop up.
2. Select Visual C# | Empty Web Site as the template, and leave the Location set as File System. Change the web site name from WebSite1 to *C:\SOAwithWCFandLINQ\Projects\HelloWorld\HostDevServer,* and click OK.


Add New Web Site
Recent Templates
Installed Templates
Visual Basic
Visual C#
Online Templates
.NET Framework 4
Sort by: Default
Search Installed Templates
ASP.NET Web Site Visual C#
ASP.NET Web Service Visual C#
Empty Web Site Visual C#
Silverlight 1.0 Web Site Visual C#
WCF Service Visual C#
ASP.NET Reports Web Site Visual C#
Dynamic Data Linq to SQL Web Site Visual C#
Dynamic Data Entities Web Site Visual C#
Type: Visual C#
An empty Web site
Browse...
OK
Cancel

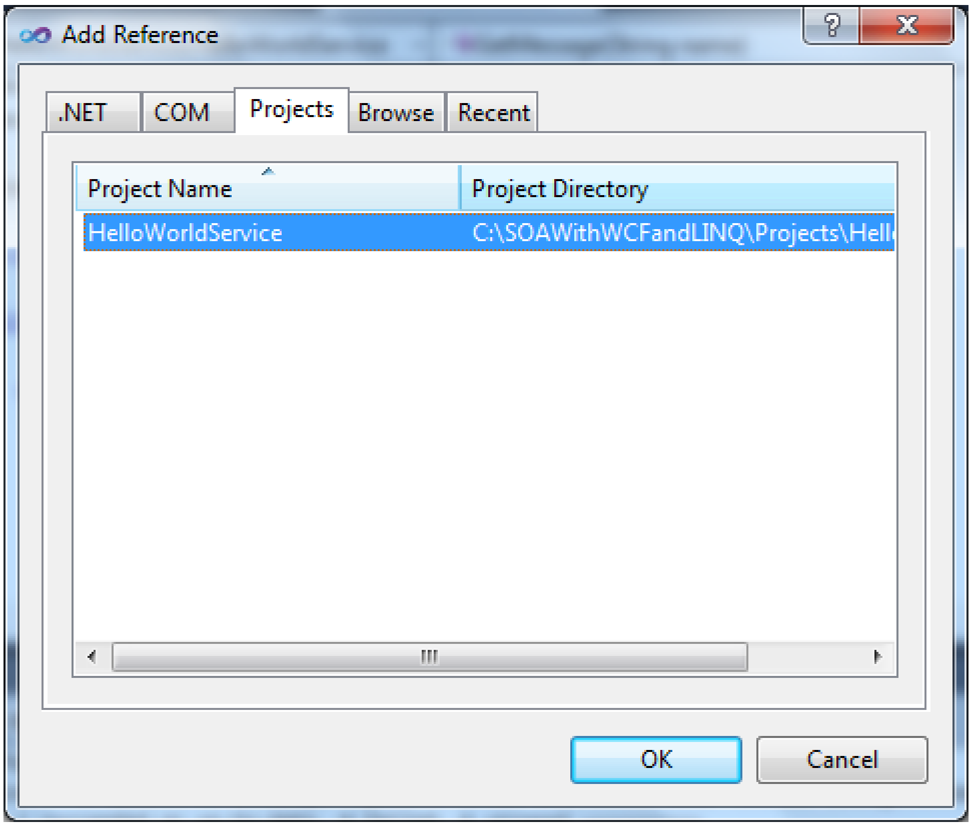
Web location: File System WCFandLINQ\Projects\HelloWorld\HostDevServer

3. Now in the Solution Explorer, you have one more item (HostDevServer) within the solution. It will look like the following:



4. Next, we need to set the website as the startup project. In the Solution Explorer, right-click on the web site*C:\...\HostDevServer*, and select Set as StartUp Project from the context menu (or you can first select the web site from the Solution Explorer, and then select the menu item Website | Set as StartUp Project). The web site*C:\...\HostDevServer* should be highlighted in the Solution Explorer, indicating that it is now the startup project.

5. Because we will host the HelloWorldService from this web site, we need to add a HelloWorldService reference to the web site. In the Solution Explorer, right-click on the web site *C:\...\HostDevServer* and select Add Reference... from the context menu. The following Add Reference dialog box should appear:

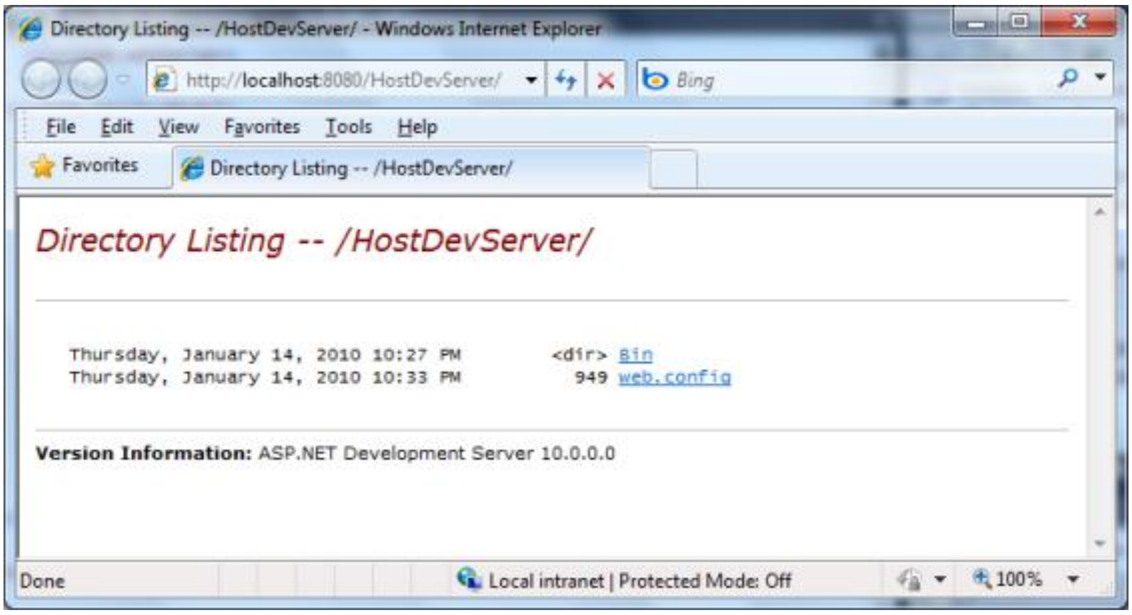


Project Name	Project Directory
HelloWorldService	C:\SOAWithWCFandLINQ\Projects\Hell

6. In the Add Reference dialog box, click on the Projects tab, select the HelloWorldService project, and then click OK. You will see that a new directory (*bin*) has been created under the HostDevServer web site, and two files from the HelloWorldService project have been copied to this new directory. Later on, when this web site is accessed, the web server (either ASP.NET Development Server, or IIS) will look for the executable code in this*bin* directory.

Testing the host application

Now we can run the website inside the ASP.NET Development Server. If you start the web site HostDevServer, by pressing Ctrl+F5, or select the Debug | Start Without Debugging... menu, you will see an empty web site in your browser. Because we have set this website as the startup project, but haven't set any start page, it lists all of the files and directories inside the *HostDevServer* directory (Directory Browsing is always enabled for a website within the ASP.NET Development Server).

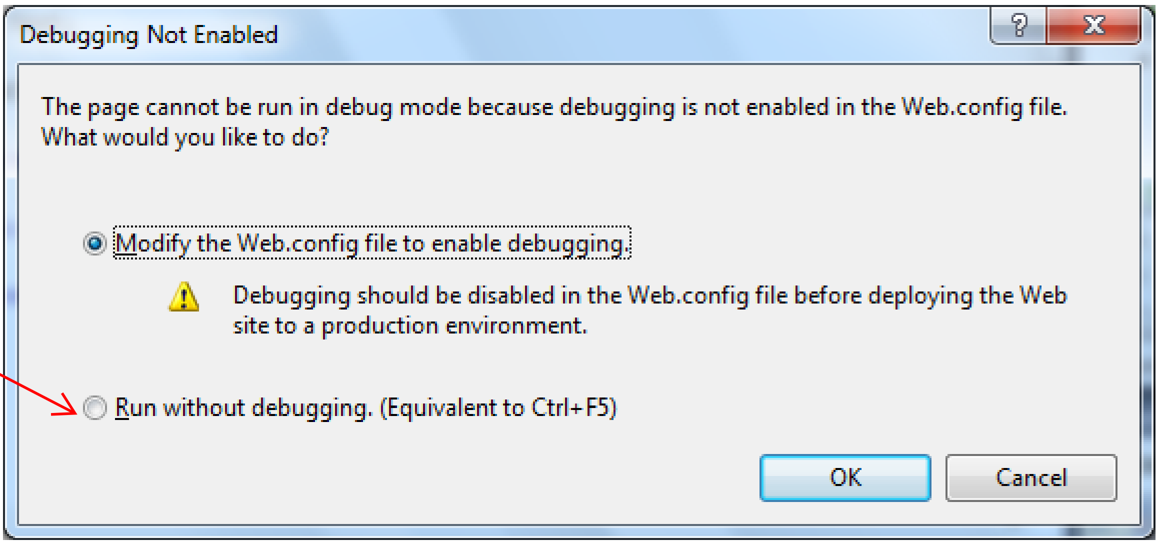

Directory Listing -- /HostDevServer/ - Windows Internet Explorer
http://localhost:8080/HostDevServer/
Bing
File Edit View Favorites Tools Help
Favorites
Directory Listing -- /HostDevServer/
Done
Local intranet | Protected Mode: Off
100%

Directory Listing -- /HostDevServer/

```
Thursday, January 14, 2010 10:27 PM        <dir> Bin
Thursday, January 14, 2010 10:33 PM          949 web.config
```

Version Information: ASP.NET Development Server 10.0.0.0

If you press F5 (or select Debug | Start Debugging from the menu), you may see a dialog saying Debugging Not Enabled (as shown below). Choose the option Run without debugging (equivalent to Ctrl+F5) and click the OK button to continue. We will explore the debugging options of a WCF Service later. Until then, we will continue to use Ctrl+F5 to start the website without debugging.


Debugging Not Enabled

The page cannot be run in debug mode because debugging is not enabled in the Web.config file. What would you like to do?

[✓ Modify the Web.config file to enable debugging.]

Debugging should be disabled in the Web.config file before deploying the Web site to a production environment.

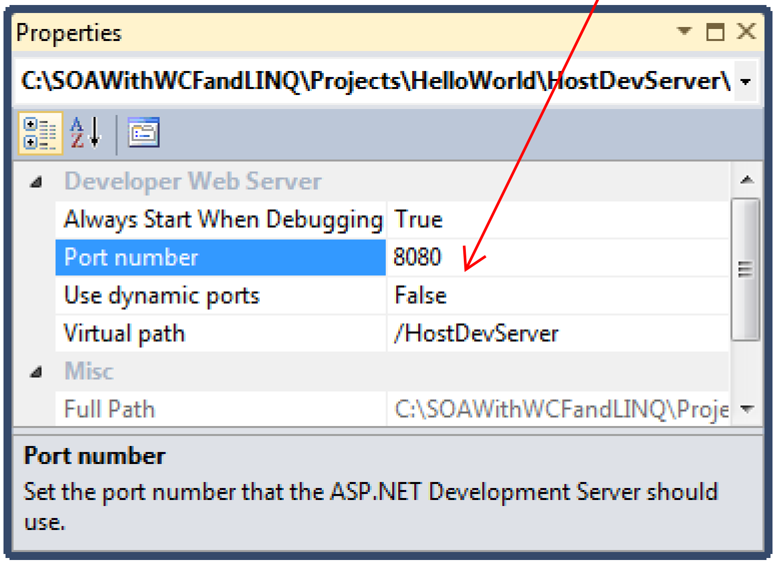
Run without debugging. (Equivalent to Ctrl+F5)

ASP.NET Development Server

At this point, you should have the HostDevServer site up and running. This site is actually running inside the built-in ASP.NET Development Server. It is a new feature that was introduced in Visual Studio 2005. This web server is intended to be used by developers only, and has functionality similar to that of the Internet Information Services (IIS) server. It also has some limitations; for example, you can run ASP.NET applications only locally. You can't use it as a real IIS server to publish a web site.

By default, the ASP.NET Development Server uses a dynamic port for the web server each time it is started. You can change it to use a static port via the Properties page of the web site. Just change the *Use dynamic ports* setting to false, and specify a static port, such as 8080, from the Properties window of the HostDevServer web site. You can't set the port to 80, because IIS is already using this port. However, if you stop your local IIS, you can set your ASP.NET Development Server to use port 80.

Note: even if you set its port to 80, it is still a local web server. It can't be accessed from outside your local PC.


Properties

C:\SOAWithWCFandLINQ\Projects\HelloWorld\HostDevServer\

Developer Web Server	
Always Start When Debugging	True
Port number	8080
Use dynamic ports	False
Virtual path	/HostDevServer
Misc	
Full Path	C:\SOAWithWCFandLINQ\Proje

Port number
Set the port number that the ASP.NET Development Server should use.

It is recommended that you use a static port so that client applications know in advance where to connect to the service. From now on, we will always use port 8080 in all of our examples.

The ASP.NET Development Server is normally started from within Visual Studio when you need to debug or unit test a web project. If you really need to start it from outside Visual Studio, you can use a command line statement in the following format:

⊟ Collapse | Copy Code

```
start /B WebDev.WebServer [/port:<port number>] /path:<physical path>
     [/vpath:<virtual path>]
```

For our web site, the statement should be like this:

⊟ Collapse | Copy Code

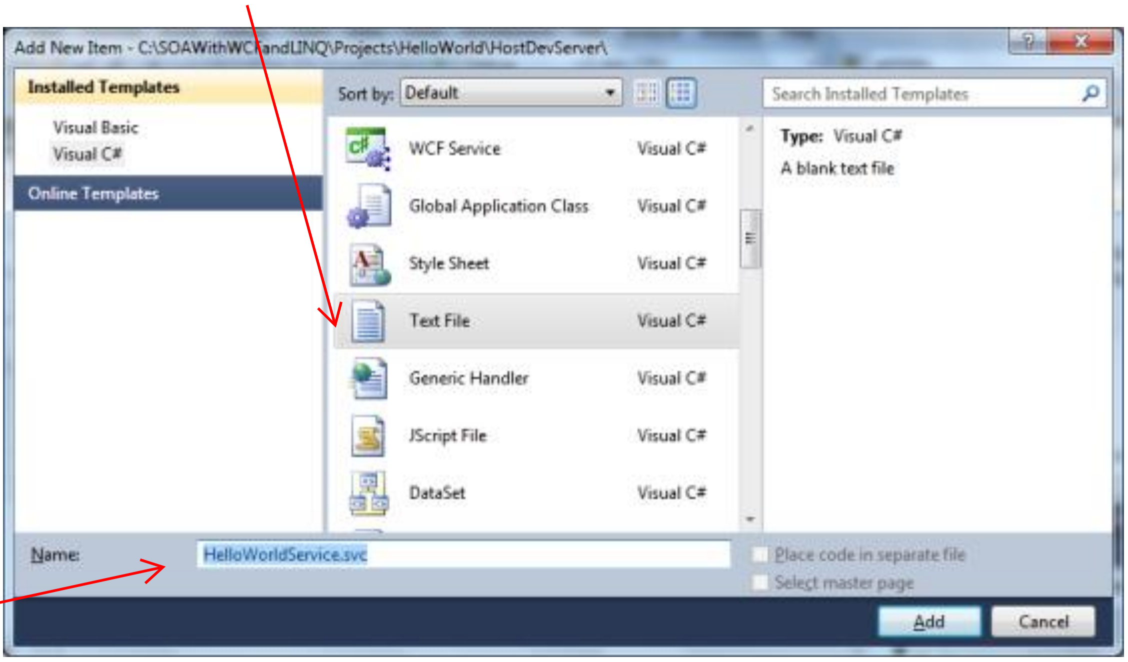
```
start /B webdev.webserver.exe /port:8080
  /path:"C:\SOAwithWCFandLINQ\Projects\HelloWorld\HostDevServer"
  /vpath:/HostDevServer
```

webdev.webserver.exe is located under your .NET framework installation directory (*C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727* or *C:\Program Files\Common Files\Microsoft Shared\DevServer*), and it may be called a different name such as *webdev.webserver20.exe*, or*webdev.webserver40.exe*.

Adding an SVC file to the host application

Although we can start the web site now, it is only an empty site. Currently, it does not host our HelloWorldService. This is because we haven't specified which service this web site should host, or an entry point for this web site. Just as an ASMX file is the entry point for a non-WCF Web Service, a *.svc* file is the entry point for a WCF Service, if it is hosted on a web server. We will now add such a file to our web site.

From the Solution Explorer, right-click on the web site *C:\...\HostDevServer*, and select Add New Item... from the context menu. The Add New Item dialog window should appear, as shown below. Select Text File as the template, and change the Name from *TextFile.txt* to *HelloWorldService.svc* in this dialog window.



Name: HelloWorldService.svc

You may have noticed that there is a template, WCF Service, in the list. We won't use it now, as it will create a new WCF Service within this web site for you (we will use this template later).

After you click the Add button in the Add New Item dialog box, an empty *svc* file will be created and added to the web site. Now enter the following line in this file:

Collapse | Copy Code

```
<%@ServiceHost Service="MyWCFServices.HelloWorldService"%>
```

Modifying the web.config file

The final step is to modify the *web.config* file of the web site. Open the *web.config* file of the web site and change it to be like this:

Collapse | Copy Code

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="false" targetFramework="4.0" />
  </system.web>
  <system.webServer>
    <modules runAllManagedModulesForAllRequests="true"/>
  </system.webServer>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="MyServiceTypeBehaviors">
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="false" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service name="MyWCFServices.HelloWorldService"
             behaviorConfiguration="MyServiceTypeBehaviors">
        <endpoint address="" binding="wsHttpBinding"
             contract="MyWCFServices.IHelloWorldService"/>
          <endpoint contract="IMetadataExchange"
             binding="mexHttpBinding" address="mex"/>
      </service>
   </services>
  </system.serviceModel>
</configuration>
```

The behavior `httpGetEnabled` is essential, because we want other applications to be able to locate the metadata of this service. Without the metadata, client applications can't generate the proxy and thus won't be able to use the service.

We use `wsHttpBinding` for this hosting, which means that it is secure (messages are encrypted while being transmitted), and transaction-aware (we will discuss this in a later chapter). However, because this is a WS-* standard, some existing applications (for example: a QA tool) may not be able to consume this service. In this case, you can change the service to use `basicHttpBinding`, which uses plain unencrypted texts when transmitting messages, and is backward compatible with traditional ASP.NET web services (ASMX Web Services).

The following is a brief explanation of the other elements in this configuration file:

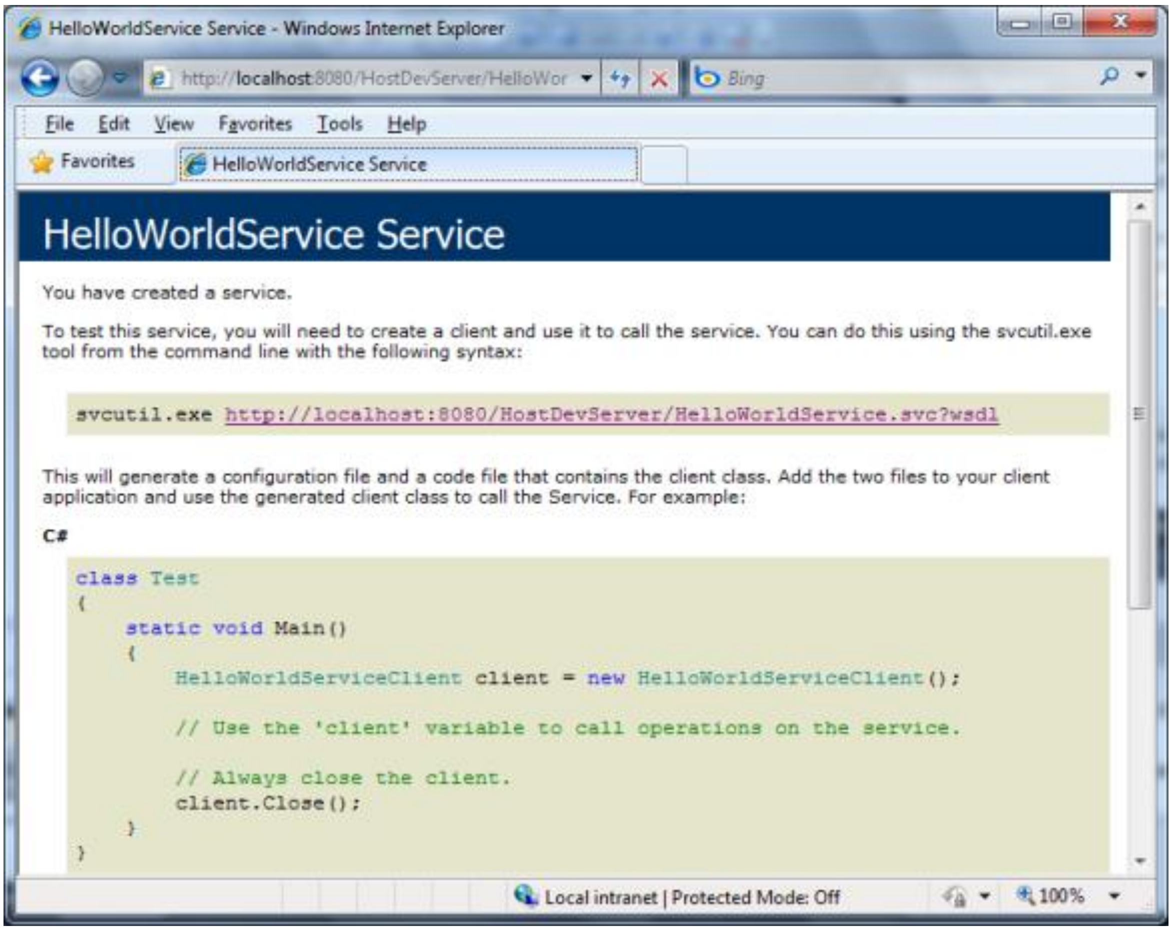
- Configuration is the root node of the file.
- `system.serviceModel` is the top node for all WCF Service specific settings.
- Within the `services` node, you can specify the WCF Services that are hosted on this web site.

In our example, we only have one WCF Service, HelloWorldService, hosted in this web site.

- Each service element defines a WCF Service, including its name, behavior, and endpoint.
- Two endpoints have been defined for HelloWorldService, one for the service itself (an application endpoint), and another for the metadata exchange (an infrastructure endpoint).
- Within the `serviceBehaviors` node, you can define the specific behaviors for a service. In our example, we have specified one behavior, which enables the service meta data exchange for the service.

Starting the host application

Now, if you start the web site by pressing Ctrl+F5 (again, don't use F5 or the menu option Debug | Start Debugging until we discuss these, later), you will find the file *HelloWorldService.svc* listed on the web page. Clicking on this file will give the description of this service, that is, how to get the WSDL file of this service, and how to create a client to consume this service. You should see a page similar to the following one. You can also set this file as the start page file so that every time you start this web site, you will go to this page directly. You can do this by right-clicking on this file in Solution Explorer and selecting Set as Start Page from the context menu.


HelloWorldService Service - Windows Internet Explorer
http://localhost:8080/HostDevServer/HelloWor
Bing
File Edit View Favorites Tools Help
Favorites
HelloWorldService Service
Local intranet | Protected Mode: Off
100%

HelloWorldService Service

You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this using the svcutil.exe tool from the command line with the following syntax:

```
svcutil.exe http://localhost:8080/HostDevServer/HelloWorldService.svc?wsdl
```

This will generate a configuration file and a code file that contains the client class. Add the two files to your client application and use the generated client class to call the Service. For example:

C#

```
class Test
{
    static void Main()
    {
        HelloWorldServiceClient client = new HelloWorldServiceClient();

        // Use the 'client' variable to call operations on the service.

        // Always close the client.
        client.Close();
    }
}
```

Now, click on the WSDL link on this page, and you will get the WSDL XML file for this service. The WSDL file gives all of the contract information for this service. In the next section, we will use this WSDL to generate a proxy for our client application.

Close the browser. Then, from the Windows system tray (systray), find the little icon labeled ASP.NET Development Server - Port 8080 (it is on the lower-right of your screen, just next to the clock), right-click on it, and select Stop to stop the service.

Creating a client to consume the WCF Service

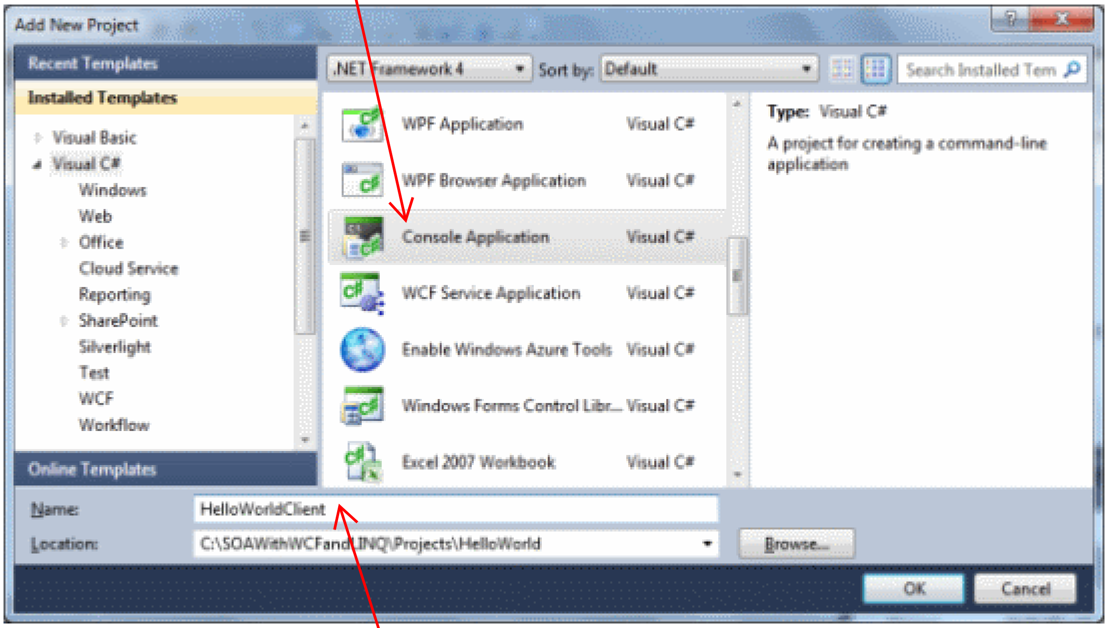
Now that we have successfully created and hosted a WCF Service, we need a client to consume the service. We will create a C# client application to consume the HelloWorldService.

In this section, we will create a Windows console application to call the WCF Service.

Creating the client application project

First, we need to create a console application project and add it to the solution. Follow these steps to create the console application:

1. In the Solution Explorer, right-click on the solution HelloWorld, and select Add | New Project... from the context menu. The Add New Project dialog window should appear, as shown below:



Add New Project
Recent Templates
Installed Templates
Visual Basic
Visual C#
Windows
Web
Office
Cloud Service
Reporting
SharePoint
Silverlight
Test
WCF
Workflow
Online Templates
WPF Application Visual C#
WPF Browser Application Visual C#
Console Application Visual C#
WCF Service Application Visual C#
Enable Windows Azure Tools Visual C#
Windows Forms Control Libr... Visual C#
Excel 2007 Workbook Visual C#
Search Installed Tem
Browse...
OK
Cancel

.NET Framework 4   Sort by: Default

Type: Visual C#

A project for creating a command-line application

Name: HelloWorldClient

Location: C:\SOAWithWCFandLINQ\Projects\HelloWorld

2. Select Visual C# | Console Application as the template; change the project name from the defaulted value of ConsoleApplication1 to HelloWorldClient, and leave the location as*C:\SOAwithWCFandLINQ\Projects\HelloWorld*. Click the OK button. The new client project has now been created and added to the solution.

Generating the proxy and configuration files

In order to consume a WCF Service, a client application must first obtain or generate a proxy class.

We also need a configuration file to specify things such as the binding of the service, the address of the service, and the contract.

To generate these two files, we can use the *svcutil.exe* tool from the command line. You can follow these steps to generate the two files:

1. Start the service by pressing Ctrl+F5 or by selecting menu option Debug | Start Without Debugging (at this point, your startup project should still be HostDevServer; if not, you need to set this to be the startup project). Now, you should see the introduction window for the HelloWorldService service, as we saw in the previous section.

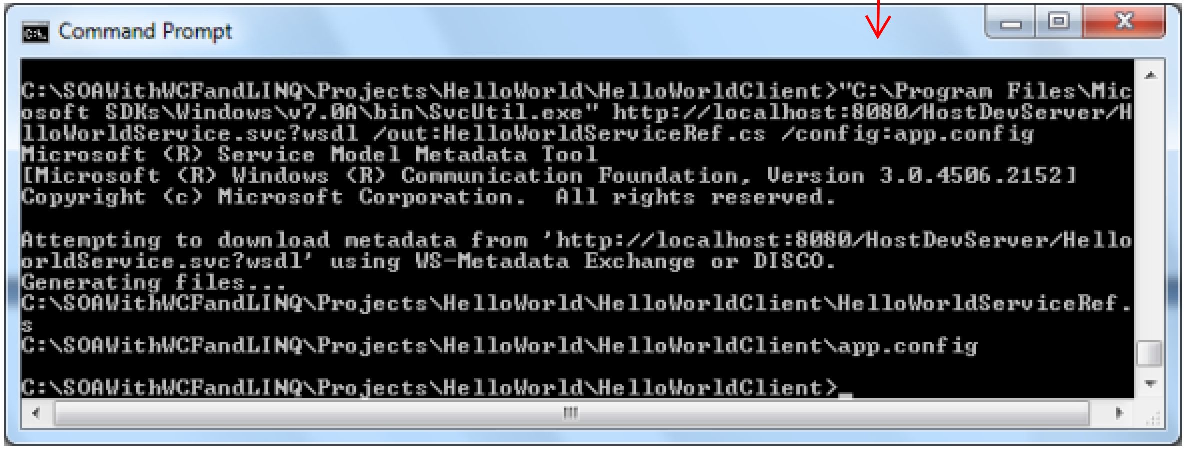
2. After the service has been started, run the command line *svcutil.exe* tool with the following syntax (*SvcUtil.exe*may be in a different directory in your machine; for example in Windows 7, it is under the *v7.0A* directory):

Collapse | Copy Code

```
C:\SOAWithWCFandLINQ\Projects\HelloWorld\HelloWorldClient>

"C:\Program Files\Microsoft SDKs\Windows\v6.0\Bin\SvcUtil.exe"
  http://localhost:8080/HostDevServer/HelloWorldService.svc?wsdl
  /out:HelloWorldServiceRef.cs /config:app.config
```

You will see output similar to that shown in the following screenshot:



```
C:\SOAWithWCFandLINQ\Projects\HelloWorld\HelloWorldClient>"C:\Program Files\Mic
osoft SDKs\Windows\v7.0A\bin\SvcUtil.exe" http://localhost:8080/HostDevServer/H
lloWorldService.svc?wsdl /out:HelloWorldServiceRef.cs /config:app.config
Microsoft (R) Service Model Metadata Tool
[Microsoft (R) Windows (R) Communication Foundation, Version 3.0.4506.2152]
Copyright (c) Microsoft Corporation.  All rights reserved.

Attempting to download metadata from 'http://localhost:8080/HostDevServer/Hello
orldService.svc?wsdl' using WS-Metadata Exchange or DISCO.
Generating files...
C:\SOAWithWCFandLINQ\Projects\HelloWorld\HelloWorldClient\HelloWorldServiceRef.
s
C:\SOAWithWCFandLINQ\Projects\HelloWorld\HelloWorldClient\app.config

C:\SOAWithWCFandLINQ\Projects\HelloWorld\HelloWorldClient>_
```

Now, two files have been generated: one for the proxy (*HelloWorldServiceRef.cs*), and the other for the configuration (*app.config*).

If you open the proxy file, you will see that the interface of the service (`IHelloWorldService`) is mimicked inside the proxy class, and a client class (`HelloWorldServiceClient`) is created to implement this interface. Inside this client class, the implementation of the service operation (`GetMessage`) is only a wrapper that delegates the call to the actual service implementation of the operation.

Inside the configuration file, you will see the definitions of the HelloWorldService, such as the endpoint address, binding, timeout settings, and security behaviors of the service.

Customizing the client application

Before we can run the client application, we have some more work to do. Follow these steps to finish the customization:

1. Adding the two generated files to the project: In Solution Explorer, click Show All Files to show all the files under the *HelloWorldClient* folder, and you will see these two files. However, they are not included in the project. Right-click on each of them and select Include In Project to include both of them in the client project. You can also use menu Project | Add Existing Item ... (or the context menu Add | Existing Item ...) to add them to the project.
2. Adding a reference to the `System.ServiceModel` namespace: Just as we did for the project HelloWorldService, we need to add a reference to the WCF .NET System.ServiceModel assembly. From the Solution Explorer, just right-click on the HelloWorldClient project, select Add Reference... and choose .NET System.ServiceModel. Then, click the OK button to add the reference to the project.
3. Modify *program.cs* to call the service: in *program.cs*, add the following line to initialize the service client object:

Collapse | Copy Code

```
HelloWorldServiceClient client = new HelloWorldServiceClient();
```

Then, we can call its method just as we would do for any other object:

⊟ Collapse | Copy Code

```
Console.WriteLine(client.GetMessage("Mike Liu"));
```

Pass your name as the parameter to the `GetMessage` method so that it prints out a message for you.

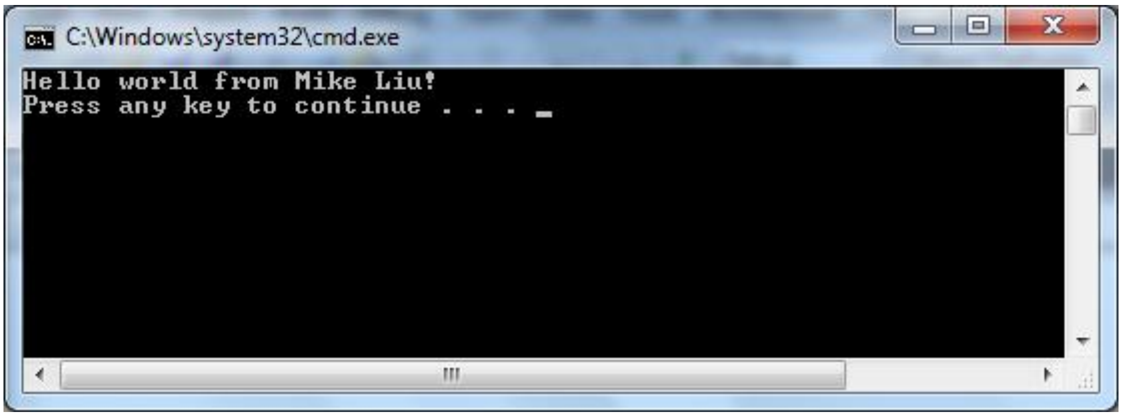
Running the client application

We are now ready to run this client program.

First, make sure the HelloWorldService has been started. If you previously stopped it, start it now (you need to set HostDevServer as the startup project, and press Ctrl+F5 to start it in non-debugging mode).

Then, from Solution Explorer, right-click on the project HelloWorldClient, select Set as StartUp Project, and then press Ctrl+F5 to run it.

You will see output as shown in the following image:

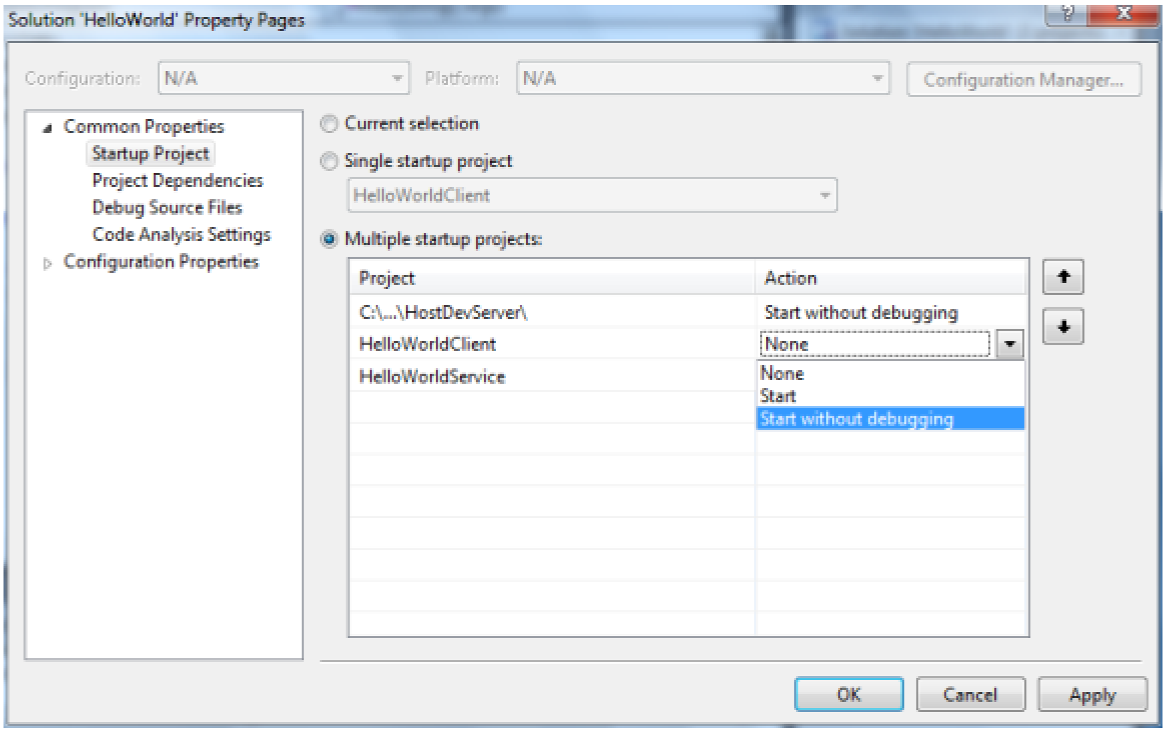


```
Hello world from Mike Liu!
Press any key to continue . . .
```

Setting the service application to AutoStart

Because we know we have to start the service before we run the client program, we can make some changes to the solution to automate this task; that is, to automatically start the service immediately before we run the client program.

To do this, in Solution Explorer, right-click on the solution, select Properties from the context menu, and you will see the Solution 'HelloWorld' Property Pages dialog box.


Solution 'HelloWorld' Property Pages

Configuration: N/A   Platform: N/A   Configuration Manager...

- Common Properties
  - Startup Project
  - Project Dependencies
  - Debug Source Files
  - Code Analysis Settings
- Configuration Properties

( ) Current selection
( ) Single startup project: HelloWorldClient
[✓ Multiple startup projects:]

Project	Action
C:\...\HostDevServer\	Start without debugging
HelloWorldClient	None
HelloWorldService	

On this page, first select the option button Multiple startup projects. Then, change the action of *C:\...\HostDevServer*\to Start without debugging. Change HelloWorldClient to the same action.

HostDevServer must be above HelloWorldClient. If it is not, use arrows to move it to the top.

To try it, first stop the service, and then press Ctrl+F5. You will notice that HostDevServer is started first, and then the client program runs without errors.

Note that this will only work inside the Visual Studio IDE. If you start the client program from Windows Explorer (*C:\SOAwithWCFandLINQ\Projects\HelloWorld\ HelloWorldClient\bin\Debug\HelloWorldClient.exe*) without first starting the service, the service won't get started automatically, and you will get an error message saying 'Could not connect to *http://localhost:8080/HostDevServer/HelloWorldService.svc*'.

Summary

In this article, we have implemented a basic WCF Service, hosted it within the ASP.NET Development Server, and created a command line program to reference and consume this basic WCF Service. At this point, you should have a thorough understanding as to what a WCF is under the hood. You will benefit from this when you develop WCF Services using Visual Studio WCF templates, or automation guidance packages. The key points covered in this chapter are:

- A WCF Service is a class library, which defines one or more WCF service interface contracts
- The System.ServiceModel assembly is referenced by all of the WCF Service projects
- The implementations of a WCF Service are just regular C# classes
- A WCF Service must be hosted in a hosting application
- Visual Studio 2010 has a built-in hosting application for WCF Services, which is called ASP.NET Development Server
- A client application uses a proxy to communicate with WCF Services
- A configuration file can be used to specify settings for WCF Services

Note: this is chapter 2 from my new book "WCF 4.0 Multi-tier Services Development with LINQ to Entities" (ISBN 1849681147). This new book is a hands-on guide to learn how to build SOA applications on the Microsoft platform using WCF and LINQ to Entities. It is updated for VS2010 from my previous book: WCF Multi-tier Services Development with LINQ.

With this new book, you can learn how to master WCF and LINQ to Entities concepts by completing practical examples and applying them to your real-world assignments. This is the first and only book to combine WCF and LINQ to Entities in a multi-tier real-world WCF Service. It is ideal for beginners who want to learn how to build scalable, powerful, easy-to-maintain WCF Services. This book is rich with example code, clear explanations, interesting examples, and practical advice. It is a truly hands-on book for C++ and C# developers.

You don't need to have any experience of WCF or LINQ to Entities to read this book. Detailed instructions and precise screenshots will guide you through the whole process of exploring the new worlds of WCF and LINQ to Entities. This book is distinguished from other WCF and LINQ to Entities books by that, this book focuses on how to do it, not why to do it in such a way, so you won't be overwhelmed by tons of information about WCF and LINQ to Entities. Once you have finished this book, you will be proud that you have been working with WCF and LINQ to Entities in the most straightforward way. You can see how much I have helped others to learn WCF and LINQ by reading the Customer Reviews on Amazon for my previous book.

You can get this book from Amazon or from the publisher's website (This book has been updated to .NET 4.5/Visual Studio 2012. You can buy this book from Amazon, or from the publisher's website athttp://www.packtpub.com/windows-communication-foundation-4-5-multi-layer-services-development-framework/book).

License