

Microsoft .NET Framework 2.0
Distributed Application Development

Mega Guide

Prepare With Confidence

This PrepLogic Mega Guide was written by certified subject matter experts and published authors to provide you accurate, in-depth exam coverage. All exam objectives are covered in detail, giving you the knowledge and confidence you need to pass your exam.



PrepLogic

Be Prepared. Be Confident. Get Certified.



Damien Foggon - Author

Create and Configure an XML Web Service

Web services are a cross platform means of exposing data and functionality to applications in a distributed environment. Web services operate over the internet using the SOAP protocol, which is based on the XML format.

Web services, under .NET, can be thought of as a normal assembly that you can interact with. The .NET runtime shields the developer from the complexities of making calls “across the wire” and appear as though they’re just a standard method call.

Create a Web Service

Visual Studio 2005 provides a project template, ASP.NET Web Service, which you can use to create an initial project; however, any ASP.NET Web project can be used to hold a Web Service.

If you select the Add New Item option for the Web project you’ll see, as shown in Figure 1, that you can add a Web Service to the project.

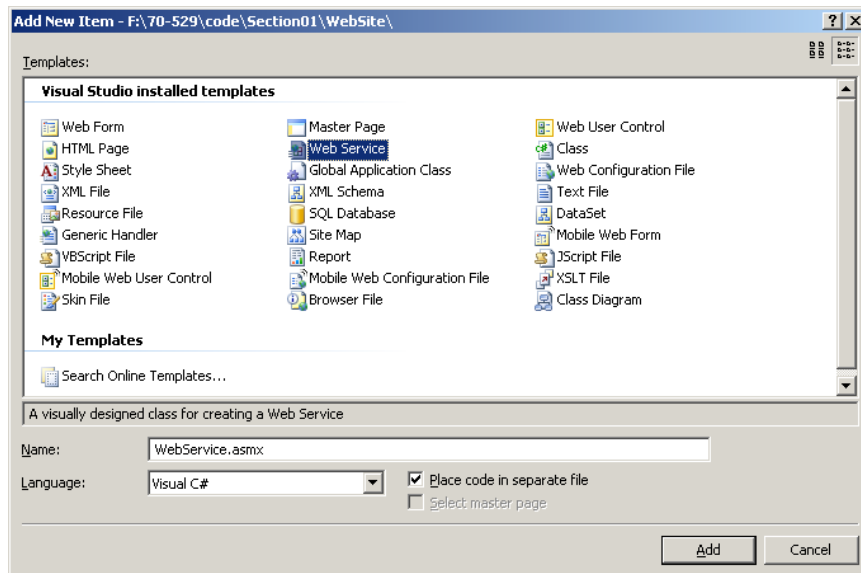


Figure 1 – Adding a Web Service

If you select the “Place code in separate file” option, referred to as the code-aside model, from the Add New Item dialog, you’ll have created two files: an ASMX file, which is the public facing for the Web Service and is equivalent to the ASPX file for Web pages, and a CS file that is added to the `App_Code` folder. This is shown in Figure 2.

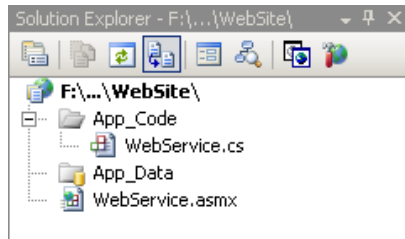


Figure 2 – The files created for a Web Service

If you've chosen not to place the code in a separate file, referred to as the code-inline model, you'll only have one file created; the ASMX file will contain everything necessary to run the Web Service.

The @WebService Directive

The first line in any ASMX file is a declaration that the file is actually a Web Service. As with the @Page directive for ASPX pages, there is a corresponding directive for Web services — @WebService. Depending on whether you've selected the code-beside or code-inline models, you'll have a slightly different syntax for the @WebService directive.

For a code-beside Web service, we specify the name of the class, the `Class` attribute, and the file that contains the class (the `CodeBehind` attribute), as follows:

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/WebService.cs"
Class="WebService" %>
```

For a code-inline Web service, we simply specify the name of the class using the `Class` attribute:

```
<%@ WebService Language="C#" Class="WebService" %>
```

Creating the Web Service Class

In order for your Web service to be compiled correctly as a Web service, there are a couple of other things that must be done:

1. The class must inherit from `System.Web.Services.WebService`.
2. The class must have the `WebService` attribute applied.

This is shown in the example below:

```
[WebService]
public class WebService : System.Web.Services.WebService
{
    // code for the class
}
```

Browsing the Web Service

Web services in ASP.NET can be added quite easily to your application. ASP.NET also provides a handy method of checking that your Web service is available and previewing the exposed methods. If you navigate to your Web service in a browser, as shown in Figure 3, you'll get a handy view of your Web service and the methods that are exposed.

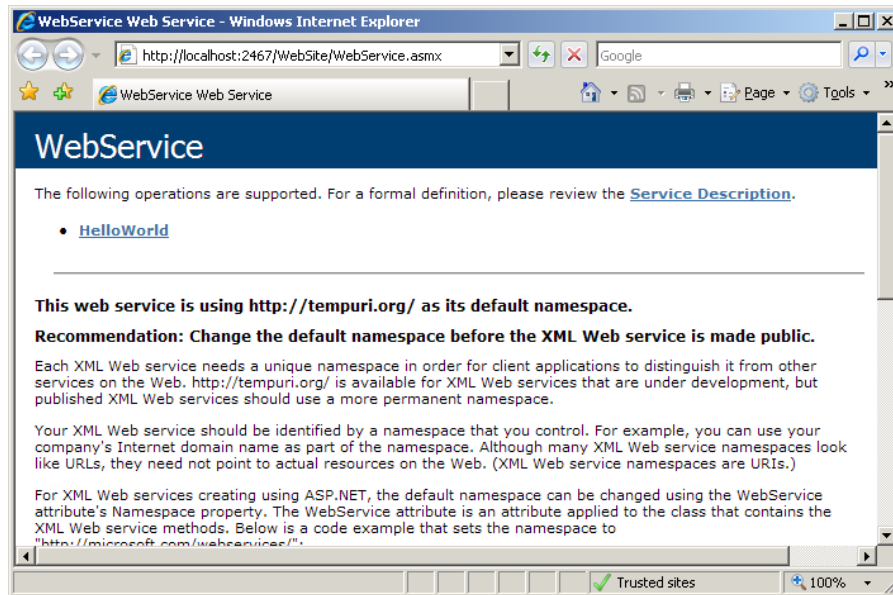


Figure 3 – Browsing a Web service

You'll see that we have a single exposed method called HelloWorld. This is added automatically to all Web services that are created in Visual Studio and the first thing that you'll normally do is delete the code for it.

Changing the Namespace

If you look again at Figure 3, you'll see that there is a recommendation to change the namespace for the Web service. All Web services created in Visual Studio are placed in this namespace. It can be changed quite easily by specifying the Namespace property of the WebService attribute:

```
[WebService(Namespace="http://preplogic.com")]  
public class WebService : System.Web.Services.WebService
```

Using the Web Service

In order to use Web services in code, you need to add a reference to it to your code. In Visual Studio there is an "Add Web Reference" attribute available for the project. Selecting this allows you to browse, as shown in Figure 4, to the required Web Service.

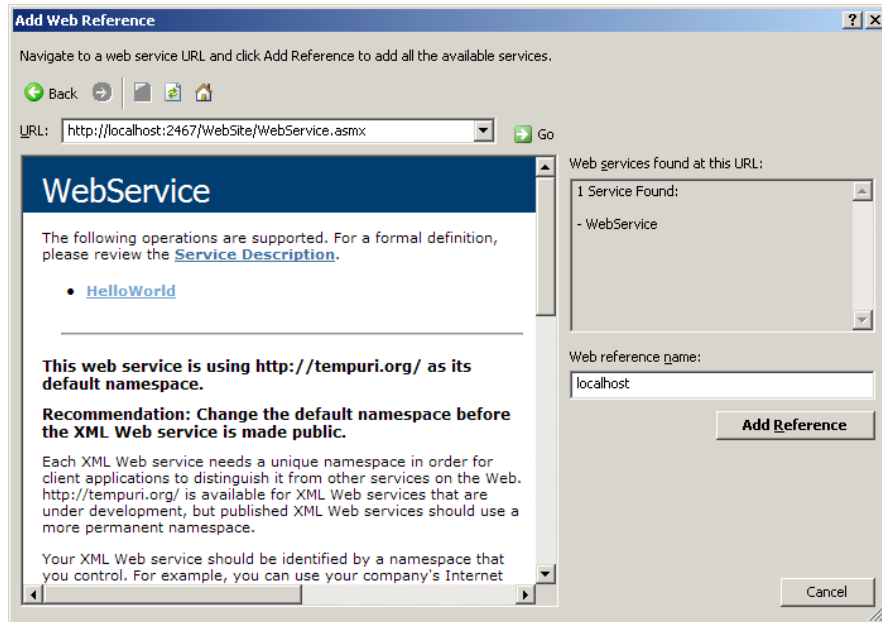


Figure 4 – Adding a reference to a Web service

The namespace box is the key; this determines the root namespace that is available within your code. In order to use the referenced Web service in code, you need to create an instance of the Web service proxy. If you leave it as localhost, as shown in Figure 4, you need to create this object as follows:

```
localhost.WebService myService = new localhost.WebService();
```

All of the exposed methods of the Web service can then be called directly on the Web service proxy. In this case we only have a HelloWorld method that returns a string, and we can call this as follows:

```
string myString = myService.HelloWorld();
```

Create Web Methods

We've already seen an example of a Web Method in the `HelloWorld` method that is added automatically to each Web service created in Visual Studio. There are only two requirements to create an exposed Web Method:

1. Create a public method in the Web service.
2. Add the `WebMethod` attribute to the method.

One of the simplest methods we can create is the aforementioned `HelloWorld` method:

```
[WebMethod]
public string HelloWorld()
{
    return "Hello World";
}
```

There are several properties that we can set within the `WebMethod` attribute. The two that you'll probably use most often are:

- `Description` – this allows you to add a description for the method. It is not available in code at the client but can provide a handy reference that is visible when viewing the Web service, such as the views we saw in Figure 3 and Figure 4.
- `MessageName` – by default, the name of the method is used as the name of the method in the proxy class. You can use the `MessageName` property to override this behavior. This is particularly handy when you have overloaded methods, as these are not supported by the SOAP protocol, and one of the overloaded methods will need to be given a different `MessageName`.

Create a OneWay Web Method

By default, when you make a call to a Web Method in your client application, the call blocks until a response is received. For the `HelloWorld` example we've seen, this is correct, as we need to return a `string`; however, for methods that don't return any values, we don't really need to wait for the method to return.

In order to mark a Web Method as *one way*, you need to use the `OneWay` property of either the `SoapDocumentMethod` or `SoapRpcMethod` attributes in the `System.Web.Services.Protocols` namespace. These two attributes change the formatting of the SOAP messages that are passed between your client and the Web service. We'll look at these two attributes in more detail later.

To add the `OneWay` attribute, you can either use the `SoapDocumentMethod` attribute:

```
[WebMethod (Namespace="http://preplogic.com")]
[SoapDocumentMethod(OneWay = true)]
public void DoWork()
```