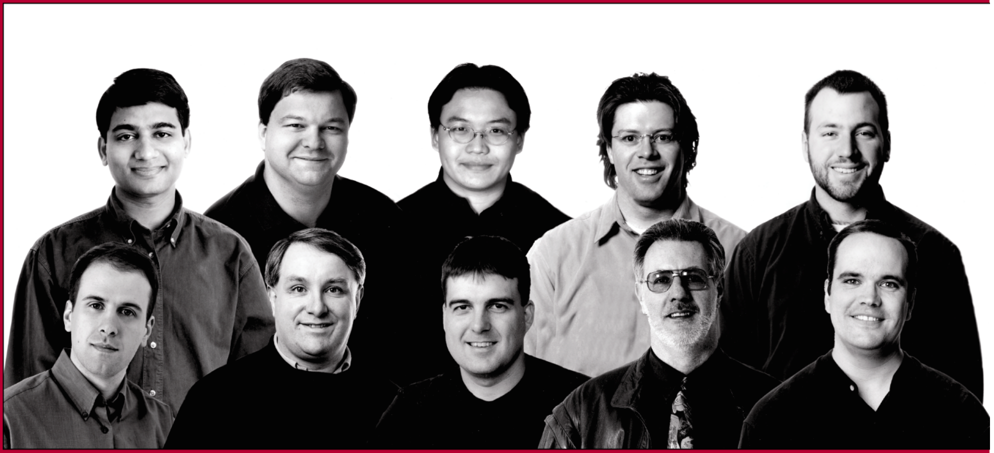
Programmer to Programmer ™

ASP.NET 1.0 with C#

Namespace Reference



Written and tested for final release of .NET v1.0

Amit Kalani, John Schenken, Bruce Lee, Matthew Gibbs, Matt Milner,
Jason Bell, Mike Clark, Andy Elmhorst, Alex Homer, Dave Gerding

wrox
Programmer to Programmer™

$39.99 USA
$54.99 CANADA

Wrox technical support at: **support@wrox.com**
Updates and source code at: **www.wrox.com**
Peer discussion at: **p2p.wrox.com**

What You Need to Use This Book

The following list is the recommended system requirements for running the code in this book:

- Windows 2000 Professional or higher with IIS installed
- Windows XP Professional with IIS installed
- ASP.NET Version 1.0
- SQL Server 2000 or MSDE
- Visual Studio .NET Professional or higher (optional)

In addition, this book assumes the following knowledge:

- A good understanding of the .NET Framework and ASP.NET
- Understanding of the C# language

Summary of Contents

9 System.Web.Services

Web Services are a new way of sharing information between computers using HTTP. The ability to create and consume Web Services, quickly and simply, is one of the evolutionary features of the .NET Framework. They can be made accessible from any point on an intranet or extranet, or over the Internet.

Not only are these services accessible over HTTP, but they also have the ability to describe themselves, what they offer, and how they can be accessed, using the emerging standards of UDDI and WSDL. Clients running on any system with support for HTTP communication can consume them, regardless of platform.

Web Services Overview

Understanding how to implement Web Services will be an important part of developing web applications in the near future. They bring a whole new suite of programming tools to classic ASP programmers, providing a mechanism for traditional enterprise programmers to expose their framework and business services in a brand new way.

The Web Services architecture will change the business model of many companies, as they will be able to easily connect to the internal systems of their suppliers, and offer direct connections for their customers to use. In the next phase of the Internet, a whole new group of client/server applications will be developed that will allow users to connect to business systems from a workstation on the Internet as seamlessly as if they were working on the local LAN.

What are Web Services?

The most basic definition of a Web Service is that it is **application logic accessible to programs via standard web protocols in a platform-independent way**. They are one way of invoking Remote Procedure Calls (RPC) over HTTP. The concept itself is nothing new, and has been implemented on network protocols many times over, using technologies like DCOM, RMI, CORBA, and EDI. What's different is that Web Services communicate using HTTP and XML, which are both industry standards.

Web Services overcome some of the limitations of previous technologies, including:

- **Platform interoperability** – Clients of Web Services do not have to be based upon any particular platform to consume the services. Platforms such as operating systems, programming languages, component specifications, or language frameworks, are irrelevant.

- **Proprietary protocols** – Web Services use the standard HTTP protocol to communicate. As such, they can easily reach through firewalls and allow communication between servers anywhere on the Internet.

It is important to understand that Web Services are more than just a single specification. They are a whole set of specifications that have been adopted by major companies such as Microsoft and IBM. These specifications will be described briefly in the section on *Key Web-Service Standards*.

Discovery and Description

Web Services are self-describing. They implement standards that allow application developers and integrators to easily find existing Web Services, discover their APIs, and build clients that can access them. The .NET Framework provides tools that can be leveraged to provide discovery and description information for the Web Services that you build.

Why Use Web Services?

There are many reasons why you may want to consider exposing Web Services using ASP.NET. We list some of the most salient points below:

Widely Accepted Standards

Web Services are based upon a set of standards that have already been adopted by a number of large software companies, and the World Wide Web Consortium (W3C). To review the latest standards affecting Web Services, go to http://www.w3.org/2002/ws/. Microsoft is seriously backing all of the latest Web Service standards, and a goldmine of developer documentation related to the subject can be found at http://msdn.microsoft.com/webservices/. A large number of tools for building Web Services have already been created, including Microsoft Visual Studio .NET, and more are being developed for other languages, such as Java and Perl.

Callable Even Through a Firewall

Communication between web clients and servers is not always smooth, often due to the settings of firewalls and proxy servers. Network administrators have very good reasons of security for not changing these settings. Nevertheless, Web Services can still expose middle-tier components as they are built on standard protocols, such as SOAP and HTTP, and they only transfer HTML and XML documents which easily pass through firewalls and proxy servers.

Cross Platform

Because Web Services are built on standard protocols, such as XML and HTTP, they are accessible from any language that can communicate over HTTP and can parse XML. A component built in Visual Basic .NET, C#, JavaScript, Perl, or Java can communicate with a component written in any other language and share data. This gives companies a new and exciting opportunity to **integrate with business partners, customers, and vendors, regardless of platform**. The servers, operating systems, and programming backgrounds of the development teams you are trying to integrate with are no longer a barrier.

Scalable

A Web Service can be deployed to run on a single web server and then later scaled across multiple servers without any changes to the application code. Better still, application updates can be made at any time without taking down the Web Service, or interrupting its servicing of clients.

Loosely Coupled

Because Web Services communicate using a message-based protocol, the service and client can evolve separately. As long as the public APIs of the Web Service do not change, recompiling one side or the other and adding functionality will not break existing installed applications.

Software as a Service

This point will perhaps be the hardest one for business managers to wrap their minds around. Building software that can be exposed to the Web allows for a whole new vertical industry that software companies can get involved in. Companies that have a valuable repository of data, information or functionality can expose it to the world using Web Services. Company-to-Company systems integration will become much more commonplace and will reach down to the level of the small business. One example of this would be a company that has a vast database of weather information. That company could expose this to the world via a Web Service. The information could then be accessed, for a fee, by media outlets such as web portals, newspapers, and other information providers. Prior to Web Services, information publishing to this broad a spectrum of customers and getting paid for it was difficult to achieve.

Reasons You May Not Want to Use Web Services

In order to balance the discussion, we'll also mention some reasons why Web Services may not be suitable for your project. They are, after all, not the answer to every application's needs.

Applications Communicating on the Same Machine or LAN

Web Services may not be well suited when different applications need to communicate with each other on the same machine or maybe even on a LAN. In this case, the performance overhead will make your application inefficient.

Remoting May Better Suit your Needs

Remoting is a feature of .NET that allows objects written in .NET languages to be hosted on a server application and accessed remotely from .NET clients. It is a very similar technology to Distributed COM (DCOM). If all of the following bullet points are true, you may be better off using .NET Remoting, instead of Web Services:

- ❑ All of the clients of your server application will be running the .NET platform
- ❑ You do not have a need for your server application to be accessible to the public on a variety of client platforms
- ❑ Performance is critical
- ❑ All of your components are on the same LAN, otherwise constant communication across firewalls becomes restrictive

This list is just a simple example; you should take the time to understand the Remoting and Web Services architectures if you are in the position of needing to make this kind of decision. A good place to find out more is in the book Professional ASP.NET Web Services, *ISBN 1-861005-458 (Wrox Press).*

Managing Access to the Web Service

Once you have published a Web Service on the Internet, it will be accessible to anyone. If it is important that only authorized users or applications have access to your Web Service, you'll have to create a way to manage access to it. This is quite easy to do with typical ASP.NET authentication and authorization mechanisms, and is mentioned here merely to remind you that it is an issue.

Key Web-Service Standards

As mentioned previously, Web Services are built upon a number of standards. The base standards of HTTP and XML are well known, and will not be covered in detail. We will look, instead, at some of the newer standards, specific to Web Services, in this section.

SOAP

The core specification for Web Service communication is **SOAP**, which stands for Simple Object Access Protocol. SOAP is a protocol for messaging between applications using XML. The SOAP standard specifies three different aspects of the messaging process:

- **Messaging Envelope** – defines an XML schema for common information that is part of every SOAP message. This information is typically referred to as an envelope because it contains addressing information and routing rules. The envelope contains a message that is being sent to, or from, the Web Service.
- **Encoding Rules** – defines how specific base data types (strings, numbers, dates, and so on) are encoded in XML.
- **Procedural conventions** – defines how an actual component method can be represented in a SOAP message packet.

A sample SOAP message, complete with HTTP Headers is shown below:

```
POST /ProgrammersReference/AddingMachine.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: nnnn
SOAPAction: "http://wrox.com/programmersreference/AddTwoNumbers"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <AddTwoNumbers xmlns="http://wrox.com/programmersreference">
      <FirstNumber>10</FirstNumber>
      <SecondNumber>5</SecondNumber>
    </AddTwoNumbers>
  </soap:Body>
</soap:Envelope>
```

In the sample, you can see a `<soap:Envelope>` element. This is the root element of the XML document, and all other elements are nested inside it according to the rules of XML. Inside the root element we have a `<soap:Body>` element that acts as a container for the message sent to, or from, the Web Service. Within this container there is a nested group of elements defining the method call. If a Web Service returns an exception, the `Body` element will contain a `Fault` element describing the exception. The .NET Framework automatically handles the creation and processing of SOAP packets. More information can be found by reading the SOAP specification on the W3C's site at http://www.w3.org/TR/SOAP.

Data Types Supported in SOAP

In order to make cross-platform integration possible, SOAP defines a base set of data types that can be used in SOAP messaging. The data types are converted to the Web Services data type system, **XSD**. XSD defines certain basic standard data types and provides the ability to define custom types. (Using the base types will make for easier integration with other systems and platforms).

Here is a list of the primitive data types that can be exposed as parameters and return values for Web Methods. They are likely to be supported on most programming platforms:

Type	Description
Simple types	Here is a list of the .NET types that are mapped to the simple types defined in the SOAP specification: `String`, `Int64`, `UInt64`, `Int32`, `UInt32`, `Int16`, `UInt16`, `Byte`, `Boolean`, `Decimal`, `Double`, `Single`, `DateTime`, and `XMLQualifiedName`
`Enumerations`	Enumerations can be built using any of the simple types defined above, except for the `Boolean` type
`Arrays`	Arrays of the above types

For Web Service applications that will be serving .NET clients exclusively, you have a lot more flexibility to use richer data types. Here is a list of some of the more common types that are specific to .NET that you can also expose as parameters from a Web Service:

Type	Description
`XmlNode`	A fragment of an XML document.
`Classes and Structs`	Any class or struct. Only the public properties and methods will be marshaled across the wire. It is also important to note that the effect is to pass the class or struct **by value**, not by reference. To use remote classes or structs by reference, you must use .NET Remoting.
`DataSet`	The ADO.NET `DataSet`.
`Arrays`	Arrays of the above types.

WSDL

WSDL stands for Web Services Description Language. It is closely tied to SOAP and is a specification for describing Web Services to developers. WSDL is a schema for an XML document that defines, in detail, the methods that are available from a specific Web Service. It also defines the SOAP messages that must be created to communicate with each Web Method. A WSDL document is commonly referred to as a **WSDL contract** because it is a documented description of a Web Service that a developer can rely upon to be accurate enough to build client access code against. ASP.NET can automatically generate WSDL for any Web Service that you create. Web clients can use the `wsdl.exe` tool to generate a proxy class by passing it the service description of the Web Service. This proxy class can then be used by client code to access the Service. The great thing about the proxy class is that it enables us to reference a remote Web Service and use its functionality within our application as if the data it returns were generated locally.

Passing WSDL as the query parameter to the Web Service can access a service description of any .NET Web Service, such as the following for example:

```
http://localhost/WebService1/AddingMachine.asmx?wsdl
```

ASP.NET Web Services take `.asmx` as their filename extension.

Here is an example piece of a WSDL contract that describes a Web Method that can add two numbers together to produce a result:

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:s="http://www.w3.org/2001/XMLSchema">
  <service name="AddingService">
    <documentation>Adding Service Web Methods</documentation>
    <port name="AddingServiceHttpGet" binding="s0:AddingServiceHttpGet">
      <http:address
            location="http://localhost/WebService1/AddingMachine.asmx" />
    </port>
  </service>
</definitions>
```

This is just a small fragment of the entire contract that must be produced in order to fully document a Web Service. Because you are programming in ASP.NET, you may never have the need to fully understand the WSDL for your Web Service, because the .NET Framework provides numerous tools that harness the power of WSDL whilst abstracting the actual implementation from view. However, as with any technology, gaining an understating of the underlying ideas always makes you a more powerful programmer.

More information about WSDL can be found in the next chapter.

UDDI

UDDI (Universal Description, Discovery and Integration) is a specification to define a way of creating a registry of the Web Services available. This allows business to publish information about the Web Services that they are offering, and also search for providers of services that they need.

UDDI defines an XML schema for providing information about Web Services. It defines a SOAP API that the UDDI registry server must implement to allow for the publishing (and querying) of Web Services. More information about UDDI, including current UDDI registries, can be found at http://www.uddi.org. Microsoft is hosting one of the first UDDI registries available, at http://uddi.microsoft.com/.

The term "UDDI" is sometimes used to refer to the registry itself, rather than the specification.

DISCO

DISCO (which is shortened from the word "discovery") is a specification that allows an XML document to be created that can be queried for the locations of WSDL documents. The first part of the specification deals with how an application can go about finding and using documents describing Web Services. The second part defines an XML schema for documenting the locations of WSDL documents.

The .NET Framework ships with a tool, `Disco.exe`, which searches for .NET Web Services located on a machine. It returns a DISCO document containing the locations of the WSDL files for those services. For more information about the discovery of Web Services, see Chapter 11.

The System.Web.Services Namespace

`System.Web.Services` contains some of the most basic classes that are used to create a Web Service. It defines the basic attributes used to describe the behavior of the Web Methods and the basic description attributes of the Web Service. Here is a list of the classes available in the `System.Web.Services` namespace.

Class	Description
`WebMethodAttribute`	**This class is a required attribute for any public method that you wish to expose as a Web Method.**
`WebService`	**This class acts as an optional base class to create a Web Service. When you inherit from this class, you have access to intrinsic ASP.NET objects.**
`WebServiceAttribute`	**This class is an optional attribute for any class that you wish to expose to the world as a Web Service. It provides additional information about the Web Service class like its name, description and namespace.**
`WebServiceBindingAttribute`	**This attribute can be applied to a class to declare that the class implements a binding defined in a WSDL contract.**

WebMethodAttribute Class

The `WebMethodAttribute` class is a required attribute for any public method that you wish to expose to the world as a Web Method. Only methods that are declared public and contain `WebMethodAttribute` will be accessible to any remote web client. When the ASP.NET runtime sees this attribute on your method, it does the necessary work of generating WSDL for it and mapping appropriate SOAP messages to it. This attribute class is derived from the `System.Attribute` class and it cannot be further inherited. All of the properties of this attribute class are optional.

WebMethodAttribute Class Public Methods

- `Equals` – inherited from `System.Object`, see Introduction for details.
- **`GetHashCode`**
- `GetType` – inherited from `System.Object`, see Introduction for details.
- **`IsDefaultAttribute`**
- **`Match`**
- `ToString` – inherited from `System.Object`, see Introduction for details.

GetHashCode

The `GetHashCode` method returns the hash code for the current instance of the object. It is inherited from `System.Attribute` and is overridden here to return the hash code.

```
public override int GetHashCode();
```

IsDefaultAttribute

The `IsDefaultAttribute` method if overridden is used to indicate whether the values of the current instance are equal to the values of the default instance of the object. It is inherited from `System.Attribute` and is not overridden in the `WebMethodAttribute` class. As it is not overridden here, it always returns `false` irrespective of the current object holding default values or not.

```
public virtual bool IsDefaultAttribute();
```

Match

The `Match` method if overridden is used to compare current instance with another object. It is inherited from `System.Attribute` and is not overridden in the `WebMethodAttribute` class. As it is not overridden here, it returns the same value that the `Equals` method would return.

```
public virtual bool Match(object obj);
```

WebMethodAttribute Class Protected Methods

- `Finalize` – inherited from `System.Object`, see Introduction for details.
- `MemberwiseClone` – inherited from `System.Object`, see Introduction for details.

WebMethodAttribute Class Public Properties

- **`BufferResponse`**
- **`CacheDuration`**
- **`Description`**
- **`EnableSession`**

- **`MessageName`**
- **`TransactionOption`**
- `TypeId` – inherited from `System.Attribute`, not implemented in this class.

BufferResponse

If this property is set to `true` (the default), the SOAP layer will create the complete response SOAP packet in a memory buffer before sending it back to the client. If it is set to `false`, the SOAP packet is sent back to the client in pieces, as it is built on the server. Normally you'll want to leave this with the default value. However, if you anticipate that the Web Method will return a very large amount of data as its result, then setting the `BufferResponse` property to `false` will improve perceived performance by disabling buffering. You should also verify when all of the data has been received.

Here is an example that sets the `BufferResponse` property to `false` as the method returns a huge `DataSet` object:

```
[WebMethod(BufferResponse=False)]
public DataSet GetData()
{
  //We turned buffering off because we are going to return lots of data
  return db.HugeDataSet();
}
```

Note that if the `BufferResponse` property is set to `false`, SOAP extensions will be disabled for this method. See Chapter 11 for more information on SOAP extensions.

```
public bool BufferResponse {get; set;}
```

CacheDuration

Web Service output caching is a feature of ASP.NET that adds virtually automatic caching capability to methods exposed by your Web Service. If a Web Method is called more than once with the same parameters before the cached output expires, ASP.NET will return the cached output without calling your method again. The data that ASP.NET caches is the SOAP message packet that was generated as a response from the previous invocation of your Web Service. By default, the `CacheDuration` property is set to 0, which means that caching is disabled. If you set the `CacheDuration` property to a number greater than 0, then ASP.NET will cache the data for that many seconds.

Here's an example of how to set up output caching of a Web Method for 60 seconds:

```
[WebMethod(CacheDuration=60)]
string GetTaskList(string GroupID)
{
  return db.getTaskList(GroupID);
}
```

A cached copy of the data will be maintained for each distinct set of parameters that are passed to the Web Method. If your Web Method will return large amounts of data and has a large number of different parameter combinations that it will be called with, caching the data may not be the right choice.

Caching frequently used pieces of data for even short periods of time can dramatically increase the potential scalability of your web application. You should weigh up the benefits against the amount of server resources you could be tying up to maintain the caches.

```
public int CacheDuration {get; set;}
```

Description

The `Description` property contains the string (default is `String.Empty`) description of the Web Method. This property is very helpful as its value is displayed in the browser when the Web Service test page is viewed in a browser. Also, this property comes in the service description document of the Web Service in the documentation element. You can help other developers understand your code by giving a brief description identifying the purpose of the method.

```
public string Description {get; set;}
```

Example: Describing Your Web Service

In this example we are going to create a simple Web service which returns the current date and time and describe it as such using the `Description` property. Here is the Web Service code, `DateTime.asmx.cs` in the code download for this chapter:

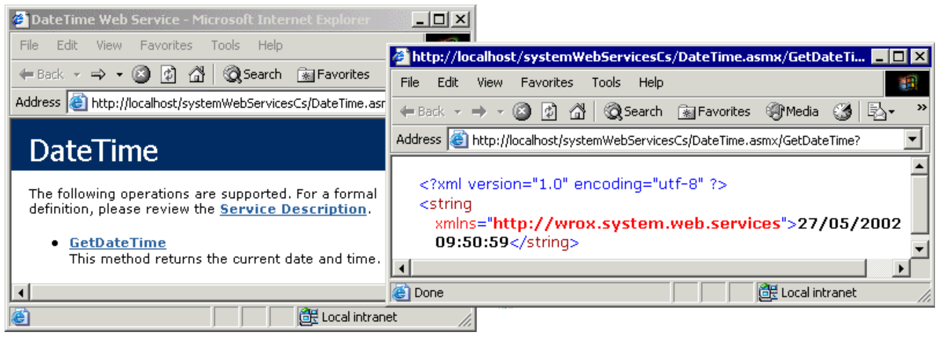
```
using System.Web.Services;

[WebService (Namespace="http://wrox.system.web.services")]
public class DateTime : System.Web.Services.WebService

...

  [WebMethod(Description="This method returns the current date and time.")]
  public string GetDateTime()
  {
    return System.DateTime.Now.ToString();
  }
```

Below-left is the output we get if we browse to the Web Service test page, `DateTime.asmx`, and below-right is the output of the Web Service if we invoke it from the default test page:


DateTime Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/systemWebServicesCs/DateTime.asr
Local intranet
http://localhost/systemWebServicesCs/DateTime.asmx/GetDateTi...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/systemWebServicesCs/DateTime.asmx/GetDateTime?
Done
Local intranet

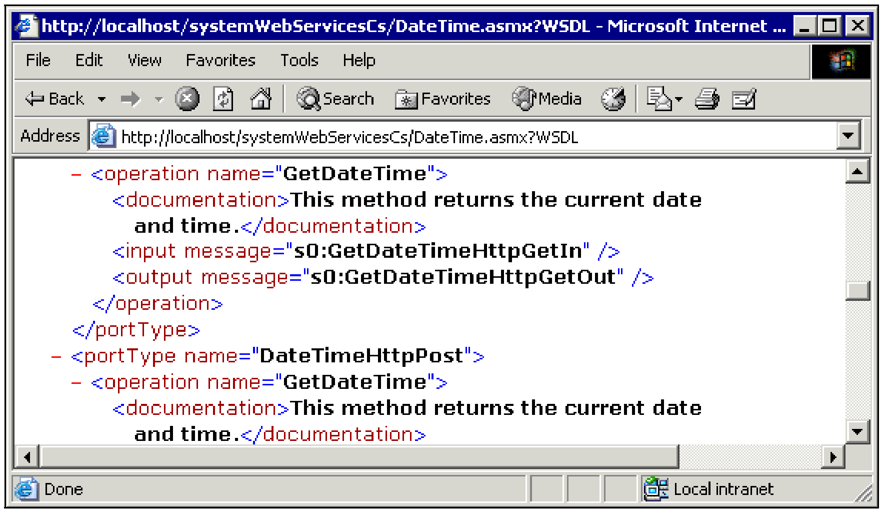
DateTime

The following operations are supported. For a formal definition, please review the Service Description.

- GetDateTime
  This method returns the current date and time.

```
<?xml version="1.0" encoding="utf-8" ?>
<string
  xmlns="http://wrox.system.web.services">27/05/2002
  09:50:59</string>
```

Furthermore here is an extract of the output if we append ?WSDL to the query string (or click the Service Description link in the test page) in order to make the ASP.NET runtime generate a WSDL contract for our Web Service:


http://localhost/systemWebServicesCs/DateTime.asmx?WSDL - Microsoft Internet ...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/systemWebServicesCs/DateTime.asmx?WSDL
Done
Local intranet

```
  - <operation name="GetDateTime">
      <documentation>This method returns the current date
        and time.</documentation>
      <input message="s0:GetDateTimeHttpGetIn" />
      <output message="s0:GetDateTimeHttpGetOut" />
    </operation>
  </portType>
- <portType name="DateTimeHttpPost">
  - <operation name="GetDateTime">
      <documentation>This method returns the current date
        and time.</documentation>
```

EnableSession

This property enables use of the Session object in a Web Method, if set to true. By default it is set to false. A Web Service can only maintain a Session object through the use of cookies. If the <sessionState> section of the web.config is set as cookieless="true", then the Web Method will not be able to maintain a Session state for the Web Method. Web Service clients that do not properly handle cookies will not have any state maintained for them in their Session objects on the server between Web Service requests, regardless of whether this property is set to true or false.

If this property is set to true and the Web Service class inherits from System.Web.Services.WebService, then the session state collection can be accessed using the WebService.Session property. If the Web Service does not inherit from the WebService class, session state can be accessed through the HttpContext.Current.Session property.

Note that setting this property to true may increase performance overhead.

```
public bool EnableSession {get; set;}
```

Example: Enabling Session State in a Web Service

The following example, `HitCounter.asmx` in the code download for this chapter, shows how to enable session state for your Web Service. Here is the code-behind for the Service, `HitCounter.asmx.cs`:

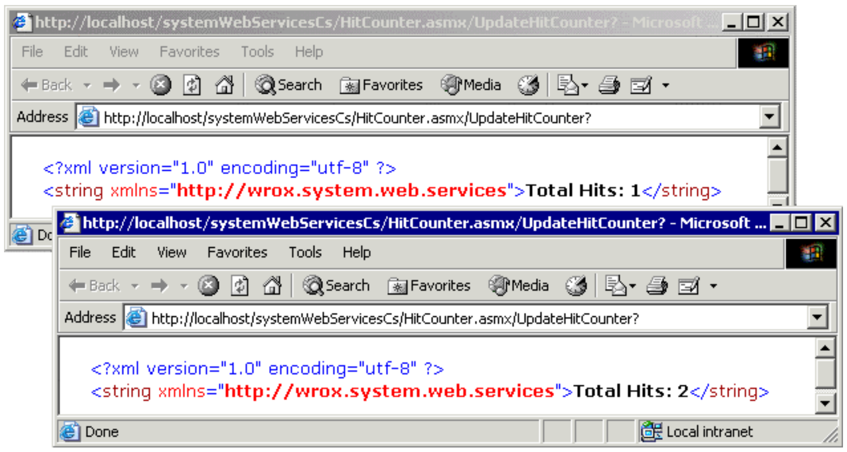
```
using System.Web.Services;

[WebService(Namespace="http://wrox.system.web.services")]
public class HitCounter : System.Web.Services.WebService

...

  [WebMethod(EnableSession=true)]
  public string UpdateHitCounter()
  {
    // Initialize Session HitCounter to 1 if it is null
    if(Session["HitCounter"] == null)
      Session["HitCounter"] = 1;
    // Increment HitCounter's value by 1
    else
      Session["HitCounter"] = (int)(Session["HitCounter"]) + 1;
    return "Total Hits: " + Session["HitCounter"].ToString();
  }
```

When you test this Web Service a couple of times you can see that the hit counter increases with each new request:



```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://wrox.system.web.services">Total Hits: 1</string>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://wrox.system.web.services">Total Hits: 2</string>
```

If you were to call this Web Method from an `.aspx` page, the web page will need to pass the Session information to the Web Method. This can be done in an `.aspx` page by creating a cookie container object and associating it with the Web Service object. Here is the piece of code that will be able to maintain session while calling a Web Method:

```
MyService  ms = new MyService();
ms.CookieContainer = new CookieContainer();
Label1.Text = ms.UpdateHitCounter();
Label2.Text = ms.UpdateHitCounter();
```

MessageName

The name that SOAP uses by default to identify your Web Method is derived from the actual method name defined in the Web Service class. This behavior can be overridden by the use of the `MessageName` property.

This property is typically used to expose Web Methods that have several overloaded versions, as SOAP does not support overloading. If your Web Service class has overloaded versions of the same method, they must be distinguished from each other by having a unique SOAP message name. Otherwise compilation errors will occur telling the user to specify unique names for methods using the `MessageName` property. If the `MessageName` property is assigned then the `name` attribute of the input and output elements of the `operation` element in the WSDL document describing the service will contain the value of the `MessageName` property.

Here is an example of an overloaded method. That is, the same name is used for two different methods, and the methods only vary by the data types of their parameters. In order for them to be exposed over SOAP as Web Methods, they must have a unique SOAP message name, and the client must reference the `MessageName` to get the correct overloaded method:

```
[WebMethod]
public string GetCustomerName(Guid CustomerID)
{
  return db.GetCustomerName(CustomerID);
}

[WebMethod(MessageName="GetCustomerByID")]
public string GetCustomerName(int CustomerID)
{
  return db.GetCustomerName(CustomerID);
}
```

The syntax for this property is as follows:

```
public string MessageName {get; set;}
```

TransactionOption

This property sets the transactional behavior for the Web Method. Enabling transactional support in a Web Method means that it can participate as the root object in a COM+ (MTS) transaction.

Web Methods in ASP.NET currently do not support sharing transactions with the application that is calling them. A Web Service is transactional only if it is the root object that started the transaction. In other words, if a Web Method begins a transaction and then calls another Web Method that requires a transaction, the two Web Services cannot mutually share the same transaction. Each Web Method participates solely in the context of its own transaction. This could change with future versions of ASP.NET as Web Service technology becomes more mature.

The TransactionOption property may be set with any of the values from the TransactionOption enumeration, which is shown below. By default, this property is set to TransactionOption.Disabled.

Note that this enumeration is from the System.EnterpriseServices namespace.

Enumeration Name	Description
Disabled	The Web Method will be run without any transaction in place.
NotSupported	Run this component outside the context of a transaction. For a Web Method, this setting is effectively the same as Disabled.
Required	If a transaction is already in place, share in it. If not, create a new transaction. For a Web Method, this is always the same as RequiresNew.
RequiresNew	Always create a new transaction for this component.
Supported	If a transaction is already in place, share in it. Because Web Methods do not share in transactions, this setting is effectively the same as Disabled.

Here is an example usage of a Web Method with transaction support enabled:

```
[WebMethod(TransactionOption=TransactionOption.RequiresNew)]
public void SaveCustomer()
{
  // run the save customer logic here within the scope of the transaction
}
```

The transaction is automatically committed unless an exception occurs in the Web Method. However, a transaction can be forcefully aborted by calling the SetAbort method of the System.EnterpriseServices.ContextUtil class.

```
public TransactionOption TransactionOption {get; set;}
```

WebService Class

WebService is a class that can be inherited to create a Web Service in ASP.NET. This class is derived from the System.ComponentModel.MarshalByValueComponent class. When you inherit from this class you get immediate access to the ASP.NET intrinsic objects like Application, Session, Server, User, Context. You should be aware that these objects are the same ones used in ASP.NET Web Form applications. In fact, the Session and Application objects are shared between Web Services and Web Forms if they are running within the same application scope in Internet Information Server (IIS).

Example: Web Service Classes

Here is an example of a Web Service class derived from `WebService` (`MyWebService.asmx.cs` in the code download):

```
using System.Web.Services;

[WebService(Namespace="http://wrox.system.web.services")]
public class MyWebService : System.Web.Services.WebService

...

[WebMethod]
public string ShowHostAddress()
{
  // Get the UserHostAddress through the
  //Context.Request.UserHostAddress property
  return Context.Request.UserHostAddress;
}
```

Note that this Web Method uses the `Context` intrinsic property, which is derived from `WebService`. The example then accesses the `UserHostAddress` property of `HttpRequest` class through the `Context` property. Properties of `WebService class` are discussed in detail later in this section.

The main advantage of inheriting from `WebService` is the intrinsic access to the `Application`, `Context`, `Server`, `Session`, and `User` objects. Because multiple-inheritance is not supported in .NET, there may be a case where you would like your class to inherit from some other class instead of `WebService`. Even if your class is not inherited from `WebService`, the intrinsic objects can still be accessed through the shared `Current` property of the `HttpContext` object of the current request. The `HttpContext` class provides access to other objects as well like `Cache`, `Request`, `Response` and so on.

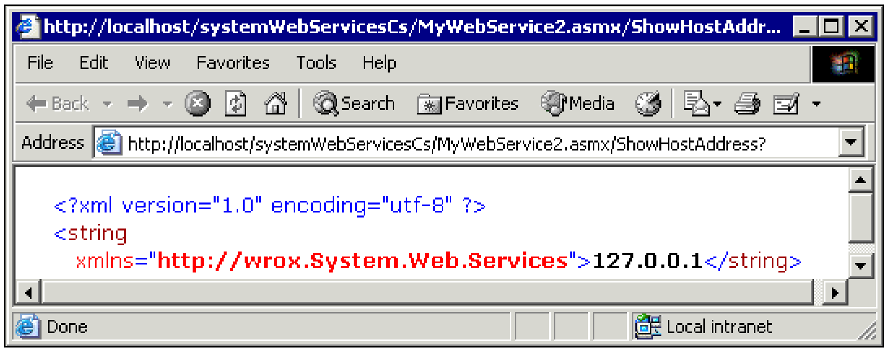
Here, the previous example is changed to provide access to the `Context` object via the `HttpContext.Current` property. Note that the Web Service also does not inherit from the `WebService` class. This code is from `MyWebService2.asmx.cs`:

```
using System.Web.Services;
using System.Web;

[WebService(Namespace="http://wrox.System.Web.Services")]
public class MyWebService2
{

  [WebMethod]
  public string ShowHostAddress()
  {
    // Get the user host address through
    // HttpContext.Current.Request.UserHostAddress
    return HttpContext.Current.Request.UserHostAddress;
  }
}
```

Both of these Web Service classes give the same output:


http://localhost/systemWebServicesCs/MyWebService2.asmx/ShowHostAddr...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/systemWebServicesCs/MyWebService2.asmx/ShowHostAddress?
Done
Local intranet

```
<?xml version="1.0" encoding="utf-8" ?>
<string
  xmlns="http://wrox.System.Web.Services">127.0.0.1</string>
```

Note that if a Web Service method applies RPC formatting via `SoapRpcMethodAttribute` or Document formatting via `SoapDocumentMethodAttribute` and sets the `OneWay` property of these classes to `true` then the Web Method will not be able to have a reference to `Context` and other intrinsic objects (`Application`, `Server`, `Session`, `User`) in any way.

WebService Class Public Methods

- **`Dispose`**
- `Equals` – inherited from `System.Object`, see Introduction for details.
- `GetHashCode`
- **`GetService`**
- `GetType` – inherited from `System.Object`, see Introduction for details.
- `ToString` – inherited from `System.Object`, see Introduction for details.

Dispose

The `Dispose` method releases all the resources used by `MarshalByValueComponent`. It is inherited from the `System.ComponentModel.MarshalByValueComponent` class.

```
public void Dispose();
```

GetHashCode

The `GetHashCode` method returns the hash code for the current instance of the object. It is inherited from `System.Attribute`.

```
public virtual int GetHashCode();
```

GetService

The `GetService` method returns the implementer of the `IServiceProvider` interface. It is inherited from the `System.ComponentModel.MarshalByValueComponent` class, and takes the type of service you want as input.

```
public virtual object GetService(Type service);
```

WebService Class Protected Methods

- **`Dispose`**
- `Finalize` – inherited from `System.Object`, see Introduction for details.
- `MemberwiseClone` – inherited from `System.Object`, see Introduction for details.

Dispose

This `Dispose` method releases all the unmanaged resources used by `MarshalByValueComponent`. It also releases the managed resources based on the `Boolean` value passed as the parameter. It is inherited from the `System.ComponentModel.MarshalByValueComponent class`.

```
protected virtual void Dispose(bool);
```

Passing in the parameter `true` indicates that the managed resources should also be released, and not if `false`. This method is called by the `Finalize` method discussed below, with `false` as the parameter value. The public `Dispose` method discussed above also calls this method with the parameter value set to `true`.

WebService Class Public Properties

- **`Application`**
- **`Container`**
- **`Context`**
- **`DesignMode`**
- **`Server`**
- **`Session`**
- **`Site`**
- **`User`**

Application

The `Application` property returns an `HttpApplicationState` object. This object is similar to the `Application` object in classic ASP.

Here is an example of using the `Application` property:

```
[WebMethod]
public string GetConnectionString()
{
  // This code will only work if we have previously stored
  // the connection string in the Application object
  return Application["ConnectStr"].ToString();
}
```

The syntax for this property is as follows:

```
public HttpApplicationState Application {get;}
```

Container

This property returns the container of the component (the implementer of the `IContainer` interface). It is inherited from the `System.ComponentModel.MarshalByValueComponent` class. This property returns `null` if the component does not have a site. `Site` is used to bind a `Component` to a `Container`.

```
public virtual IContainer  Container {get;}
```

Context

The `Context` property returns the `HttpContext` object for the current request. The `HttpContext` class contains accessors for all of the normal objects you would use to retrieve the state of the current request, including the `Session`, `Request`, `Response`, `Application`, `Server`, and `User` objects. Here is an example of getting the `Items` property of the `HttpContext` object:

```
// This code will only work if we have previously stored
// ItemKey in the Context object
string aContextItem  = Context.Items["ItemKey"].ToString();
```

The syntax for this property is as follows:

```
public HttpContext Context {get;}
```

DesignMode

This property indicates whether the container is in design mode. It is inherited from `System.ComponentModel.MarshalByValueComponent` class. This property returns `false` if the component does not have a site or is not in design mode.

```
public virtual bool DesignMode {get;}
```

Server

The `Server` property returns the current `HttpServerUtility` object. The `HttpServerUtility` object is similar to the `Server` object in classic ASP.

```
    public HttpServerUtility Server {get;}
```

Session

The `Session` property returns an `HttpSessionState` object. This object stores state for the current user between requests. The `Session` object is turned off by default in a Web Service. In order to enable it, you will need to set the `EnableSession` property on the `WebMethod` attribute to `true`. See the `WebMethodAttribute` class reference earlier in this chapter for more details.

Typically the `Session` object is not used in Web Services, as they are best designed to run in a stateless manner. In other words, each request will have all of the information the server needs to process the information and return a response. The server should not need to persist any information about the client in between requests. In some cases, however, session state within a Web Service may be a necessary and an important part of the design.

Note that the use of session state requires that the client application be able to process and return cookies for each subsequent request.

```
    public HttpSessionState Session {get;}
```

Site

This property returns the site of the component (implementer of the `ISite` interface) indicating that a component is to be added to the container. It is inherited from the `System.ComponentModel.MarshalByValueComponent` class. This property returns `null` if the component has not been added to the container, or is being removed from the container. Nevertheless this property does not remove a component from a container.

```
    public virtual ISite Site {get; set;}
```

User

If user authentication has been enabled for the current Web Service, the `User` property will contain an object that implements the `IPrincipal` interface and represents the identity of the current user. See Chapter 8 for more details on user authentication. Here is an example of retrieving the current user's name.

```
    string aName  = User.Identity.Name;
```

The syntax for this property is as follows:

```
    public IPrincipal User {get;}
```

WebService Class Protected Properties

- ❑ **`Events`**

Events

This property returns an `EventHandlerList` object representing all the event handlers attached to this component. If there are any event handlers associated with the component, they will be accessible through the `EventHandlerList` object. You can call various methods of the `EventHandlerList` object on the return value. It is inherited from the `System.ComponentModel.MarshalByValueComponent` class.

```
protected EventHandlerList Events {get;}
```

WebService Class Public Events

- ❑ **`Disposed`**

Disposed

The `Disposed` event can be used to add an event handler. This event handler is invoked whenever the `Disposed` event occurs. It is inherited from the `System.ComponentModel.MarshalByValueComponent` class.

```
   public Event EventHandler Disposed;
```

WebServiceAttribute Class

The `WebServiceAttribute` class is an optional attribute for any class that you wish to expose to the world as a Web Service. It provides additional information about the Web Service class including the custom name, description of the Web Service and most importantly the namespace to which the Web Service belongs. It is good programming practice to specify all three of these properties when your Web Service is ready to be exposed to the outside world. They provide great information about the Web Service to web clients who may want to consume it. This attribute class is derived from the `System.Attribute` class and it cannot be further inherited. All of the properties of this attribute class are optional. These properties can partially change the SOAP and WSDL that is automatically generated for your code by ASP.NET at run time. The methods discussed below will not be of much significance to programmers as this class is basically used to set the behavior of the Web Method through its properties.

WebServiceAttribute Public Methods

- ❑ `Equals` – inherited from `System.Object`, see Introduction for details.
- ❑ **`GetHashCode`**
- ❑ `GetType` – inherited from `System.Object`, see Introduction for details.
- ❑ **`IsDefaultAttribute`**
- ❑ **`Match`**
- ❑ `ToString` – inherited from `System.Object`, see Introduction for details.

GetHashCode

The `GetHashCode` method returns the hash code for the current instance of the object. It is inherited from `System.Attribute` and is overridden here to return the hash code.

```
public override int GetHashCode();
```

IsDefaultAttribute

The `IsDefaultAttribute` method if overridden is used to indicate whether the values of the current instance are equal to the values of the default instance of the object. It is inherited from `System.Attribute` and is not overridden in the `WebServiceAttribute` class. As it is not overridden here, it always returns `false` irrespective of whether or not the current object is holding default values.

```
public virtual bool IsDefaultAttribute();
```

Match

The `Match` method if overridden is used to compare the current instance with another object. It is inherited from `System.Attribute` and is not overridden in the `WebServiceAttribute` class. As it is not overridden here, it returns the same value that the `Equals` method would return.

```
public virtual bool Match(object obj);
```

WebServiceAttribute Class Protected Methods

- `Finalize` – inherited from `System.Object`, see Introduction for details.
- `MemberwiseClone` – inherited from `System.Object`, see Introduction for details.

WebServiceAttribute Class Public Properties

- **`Description`**
- **`Name`**
- **`Namespace`**
- `TypeId` – inherited from `System.Attribute`, not implemented in this class.

Description

The `Description` property of the `WebServiceAttribute` can be set on your class to briefly describe your Web Service to consumers of your class. The WSDL that is generated will contain this description in a new element `documentation` added to the `service` element. The default ASP.NET Web Service description page for your Web Service also displays the description right after the header of the page. It is very similar to its `WebMethodAttribute` counterpart except its purpose is to explain the Service in general rather than a particular Web Method.

```
public string Description {get; set;}
```

Name

The `Name` property is used to assign a custom name to the Web Service. By default this property holds the name of the Web Service class. The ASP.NET Web Service description page for your Web Service also displays the Web Service name as the header of the page.

```
public string Name {get; set;}
```

Namespace

The `Namespace` property is probably the most important attribute to use. It defines a unique URI that defines the XML namespace that will be applied to the specific XML elements in SOAP messages sent to, and from, your Web Service. Your namespace is what sets your Web Service apart from all other Web Services in the world. Other Web Services could have the same method names as yours, but if your namespace is unique, there is no ambiguity about which Web Service belongs to you. Typically, businesses that already have a unique domain name for their web site will use it for part of the namespace for their Web Services.

```
public string Namespace {get; set;}
```

Example: Using Web Service Attributes

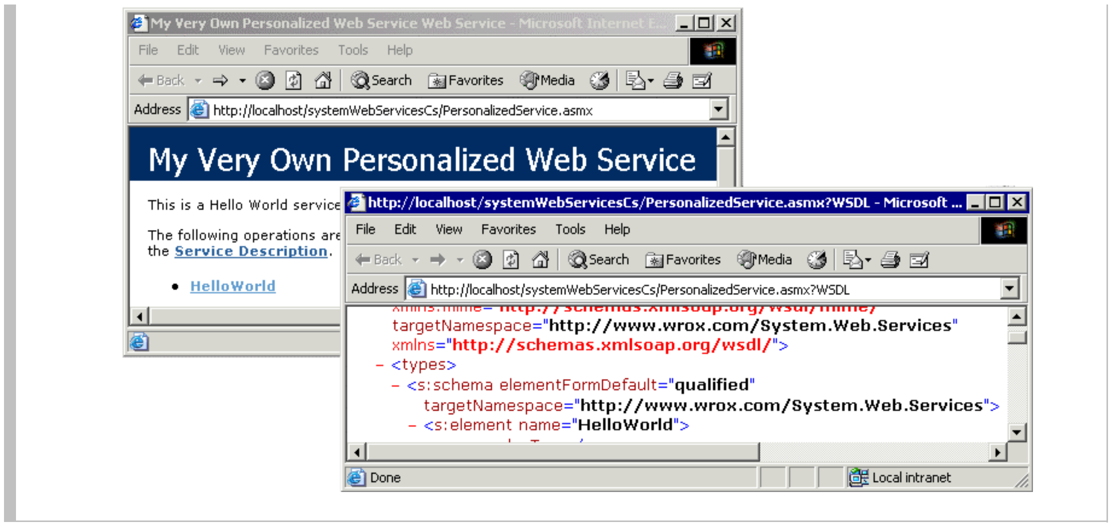
In this example we are going to use the properties we have been discussing from the `WebServiceAttribute` class to distinguish our simple Web Service. Here is the code-behind for our service, `PersonalizedService.asmx.cs`:

```
using System.Web.Services;

[WebService(Name="My Very Own Personalized Web Service",
Description="This is a Hello World service for illustration purposes",
Namespace="http://www.wrox.com/System.Web.Services")]
public class PersonalizedService : System.Web.Services.WebService
{
...

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

As you can see if we open up the test page for this service, and if we look at the WSDL contract by clicking the Service Description link, adding these attributes has given our simple Web Service a much more descriptive interface:


My Very Own Personalized Web Service Web Service - Microsoft Internet E...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/systemWebServicesCs/PersonalizedService.asmx
http://localhost/systemWebServicesCs/PersonalizedService.asmx?WSDL - Microsoft ...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/systemWebServicesCs/PersonalizedService.asmx?WSDL
Done
Local intranet

My Very Own Personalized Web Service

This is a Hello World service

The following operations are
the **Service Description**.

- **HelloWorld**

```
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  targetNamespace="http://www.wrox.com/System.Web.Services"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
- <types>
  - <s:schema elementFormDefault="qualified"
      targetNamespace="http://www.wrox.com/System.Web.Services">
    - <s:element name="HelloWorld">
```

WebServiceBindingAttribute Class

The binding defines a set of operations provided by your Web Service; you can think of this as a template or an interface. Here each operation is nothing but a Web Method. This attribute can be applied to a class to declare that the class implements a binding defined in a WSDL contract. If you wish to implement multiple bindings, you can include numerous `WebServiceBindingAttributes` in the Web Service class declaration. This attribute class is derived from the `System.Attribute` class and it cannot be further inherited. The `WebServiceBindingAttribute` simply describes the details of one or more bindings that are described elsewhere in addition to their own default bindings. For more information on bindings, see Chapter 10.

As Web Services evolve, the ability to be able to bind to a particular set of operations will become crucial to allow Web Services to seamlessly interact with one another. It is likely that standards bodies will define particular bindings for common types of Web Services that Web Service developers can then build to.

Note that if you have applied a `WebServiceBindingAttribute` attribute to your Web Service, the individual methods that implement the binding interface must also be marked with a `SoapDocumentMethodAttribute` or a `SoapRpcMethodAttribute`. These attributes map the specific Web Methods to the associated binding that they are implementing. See Chapter 11 for more details on these attributes.

WebServiceBindingAttribute Class Public Methods

- ❑ `Equals` – inherited from `System.Object`, see Introduction for details.
- ❑ **`GetHashCode`**
- ❑ `GetType` – inherited from `System.Object`, see Introduction for details.
- ❑ **`IsDefaultAttribute`**
- ❑ **`Match`**
- ❑ `ToString` – inherited from `System.Object`, see Introduction for details.

GetHashCode

The `GetHashCode` method returns the hash code for the current instance of the object. It is inherited from `System.Attribute` and is overridden here to return the hash code.

```
public override int GetHashCode();
```

IsDefaultAttribute

The `IsDefaultAttribute` method if overridden is used to indicate whether the values of the current instance are equal to the values of the default instance of the object. It is inherited from `System.Attribute` and is not overridden in the `WebServiceAttribute` class. As it is not overridden here, it always returns `false` irrespective of whether or not the current object is holding default values.

```
public virtual bool IsDefaultAttribute();
```

Match

The `Match` method if overridden is used to compare the current instance with another object. It is inherited from `System.Attribute` and is not overridden in the `WebServiceAttribute` class. As it is not overridden here, it returns the same value that the `Equals` method would return.

```
public virtual bool Match(object obj);
```

WebServiceBindingAttribute Class Protected Methods

- `Finalize` – inherited from `System.Object`, see Introduction for details.
- `MemberwiseClone` – inherited from `System.Object`, see Introduction for details.

WebServiceBindingAttribute Class Public Properties

- **`Location`**
- **`Name`**
- **`Namespace`**
- `TypeId` – inherited from `System.Attribute`, not implemented in this class.

Location

The `Location` property specifies the URL to the WSDL contract that this Web Service is bound to. The default value is the URL of the Web Service that this attribute is applied to.

```
public string Location {get; set;}
```

Name

The `Name` property defines a unique name that distinguishes this binding from all other bindings. The default value is the name of the class with the word SOAP appended to it.

```
public string Name {get; set;}
```

Namespace

The `Namespace` property sets the namespace that is associated with the particular binding.

```
public string Namespace {get; set;}
```