This page is specific to

Microsoft Visual Studio 2010/.NET Framework 4

**Comparing ASP.NET Web Services to WCF Based on Development**

Windows Communication Foundation (WCF) has an ASP.NET compatibility mode option to enable WCF applications to be programmed and configured like ASP.NET Web services, and mimic their behavior. The following sections compare ASP.NET Web services and WCF based on what is required to develop applications using both technologies.

**Data Representation**

The development of a Web service with ASP.NET typically begins with defining any complex data types the service is to use. ASP.NET relies on the [XmlSerializer](http://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer.aspx) to translate data represented by .NET Framework types to XML for transmission to or from a service and to translate data received as XML into .NET Framework objects. Defining the complex data types that an ASP.NET service is to use requires the definition of .NET Framework classes that the **XmlSerializer** can serialize to and from XML. Such classes can be written manually, or generated from definitions of the types in XML Schema using the command-line XML Schemas/Data Types Support Utility, xsd.exe.

The following is a list of key issues to know when defining .NET Framework classes that the **XmlSerializer** can serialize to and from XML:

* Only the public fields and properties of .NET Framework objects are translated into XML.
* Instances of collection classes can be serialized into XML only if the classes implement either the [IEnumerable](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx) or [ICollection](http://msdn.microsoft.com/en-us/library/system.collections.icollection.aspx) interface.
* Classes that implement the [IDictionary](http://msdn.microsoft.com/en-us/library/system.collections.idictionary.aspx) interface, such as [Hashtable](http://msdn.microsoft.com/en-us/library/system.collections.hashtable.aspx), cannot be serialized into XML.
* The great many attribute types in the [System.Xml.Serialization](http://msdn.microsoft.com/en-us/library/system.xml.serialization.aspx) namespace can be added to a .NET Framework class and its members to control how instances of the class are represented in XML.

WCF application development usually also begins with the definition of complex types. WCF can be made to use the same .NET Framework types as ASP.NET Web services.

The WCF [DataContractAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.datacontractattribute.aspx) and [DataMemberAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.datamemberattribute.aspx) can be added to .NET Framework types to indicate that instances of the type are to be serialized into XML, and which particular fields or properties of the type are to be serialized, as shown in the following sample code.

//Example One:

[DataContract]

public class LineItem

{

[DataMember]

public string ItemNumber;

[DataMember]

public decimal Quantity;

[DataMember]

public decimal UnitPrice;

}

//Example Two:

public class LineItem

{

[DataMember]

private string itemNumber;

[DataMember]

private decimal quantity;

[DataMember]

private decimal unitPrice;

public string ItemNumber

{

get

{

return this.itemNumber;

}

set

{

this.itemNumber = value;

}

}

public decimal Quantity

{

get

{

return this.quantity;

}

set

{

this.quantity = value;

}

}

public decimal UnitPrice

{

get

{

return this.unitPrice;

}

set

{

this.unitPrice = value;

}

}

}

//Example Three:

public class LineItem

{

private string itemNumber;

private decimal quantity;

private decimal unitPrice;

[DataMember]

public string ItemNumber

{

get

{

return this.itemNumber;

}

set

{

this.itemNumber = value;

}

}

[DataMember]

public decimal Quantity

{

get

{

return this.quantity;

}

set

{

this.quantity = value;

}

}

[DataMember]

public decimal UnitPrice

{

get

{

return this.unitPrice;

}

set

{

this.unitPrice = value;

}

}

}

The **DataContractAttribute** signifies that zero or more of a type’s fields or properties are to be serialized, while the **DataMemberAttribute** indicates that a particular field or property is to be serialized. The **DataContractAttribute** can be applied to a class or structure. The **DataMemberAttribute** can be applied to a field or a property, and the fields and properties to which the attribute is applied can be either public or private. Instances of types that have the **DataContractAttribute** applied to them are referred to as data contracts in WCF. They are serialized into XML using [DataContractSerializer](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.datacontractserializer.aspx).

The following is a list of the important differences between using the **DataContractSerializer** and using the **XmlSerializer** and the various attributes of the **System.Xml.Serialization** namespace.

* The **XmlSerializer** and the attributes of the **System.Xml.Serialization** namespace are designed to allow you to map .NET Framework types to any valid type defined in XML Schema, and so they provide for very precise control over how a type is represented in XML. The **DataContractSerializer**, **DataContractAttribute** and **DataMemberAttribute** provide very little control over how a type is represented in XML. You can only specify the namespaces and names used to represent the type and its fields or properties in the XML, and the sequence in which the fields and properties appear in the XML:

[DataContract(

Namespace="urn:Contoso:2006:January:29",

Name="LineItem")]

public class LineItem

{

[DataMember(Name="ItemNumber",IsRequired=true,Order=0)]

public string itemNumber;

[DataMember(Name="Quantity",IsRequired=false,Order = 1)]

public decimal quantity;

[DataMember(Name="Price",IsRequired=false,Order = 2)]

public decimal unitPrice;

}

Everything else about the structure of the XML used to represent the .NET type is determined by the **DataContractSerializer**.

* By not permitting much control over how a type is to be represented in XML, the serialization process becomes highly predictable for the **DataContractSerializer**, and, thereby, easier to optimize. A practical benefit of the design of the **DataContractSerializer** is better performance, approximately ten percent better performance.
* The attributes for use with the **XmlSerializer** do not indicate which fields or properties of the type are serialized into XML, whereas the **DataMemberAttribute** for use with the **DataContractSerializer** shows explicitly which fields or properties are serialized. Therefore, data contracts are explicit contracts about the structure of the data that an application is to send and receive.
* The **XmlSerializer** can only translate the public members of a .NET object into XML, the **DataContractSerializer** can translate the members of objects into XML regardless of the access modifiers of those members.
* As a consequence of being able to serialize the non-public members of types into XML, the **DataContractSerializer** has fewer restrictions on the variety of .NET types that it can serialize into XML. In particular, it can translate into XML types like **Hashtable** that implement the **IDictionary** interface. The **DataContractSerializer** is much more likely to be able to serialize the instances of any pre-existing .NET type into XML without having to either modify the definition of the type or develop a wrapper for it.
* Another consequence of the **DataContractSerializer** being able to access the non-public members of a type is that it requires full trust, whereas the **XmlSerializer** does not. The Full Trust code access permission give complete access to all resources on a machine that can be access using the credentials under which the code is executing. This options should be used with care as fully trusted code accesses all resources on your machine.
* The **DataContractSerializer** incorporates some support for versioning:
  + The **DataMemberAttribute** has an [IsRequired](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.datamemberattribute.isrequired.aspx) property that can be assigned a value of false for members that are added to new versions of a data contract that were not present in earlier versions, thereby allowing applications with the newer version of the contract to be able to process earlier versions.
  + By having a data contract implement the [IExtensibleDataObject](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.iextensibledataobject.aspx) interface, one can allow the **DataContractSerializer** to pass members defined in newer versions of a data contract through applications with earlier versions of the contract.

Despite all of the differences, the XML into which the **XmlSerializer** serializes a type by default is semantically identical to the XML into which the **DataContractSerializer** serializes a type, provided the namespace for the XML is explicitly defined. The following class, which has attributes for use with both of the serializers, are translated into semantically identical XML by the **XmlSerializer** and by the **DataContractAttribute**:

[Serializable]

[XmlRoot(Namespace="urn:Contoso:2006:January:29")]

[DataContract(Namespace="urn:Contoso:2006:January:29")]

public class LineItem

{

[DataMember]

public string ItemNumber;

[DataMember]

public decimal Quantity;

[DataMember]

public decimal UnitPrice;

}

The Windows software development kit (SDK) includes a command-line tool called the [ServiceModel Metadata Utility Tool (Svcutil.exe)](http://msdn.microsoft.com/en-us/library/aa347733.aspx).Like the xsd.exe tool used with ASP.NET Web services, Svcutil.exe can generate definitions of .NET data types from XML Schema. The types are data contracts if the **DataContractSerializer** can emit XML in the format defined by the XML Schema; otherwise, they are intended for serialization using the **XmlSerializer**. The tool, Svcutil.exe, can also be made to generate XML Schema from data contracts using its **/dataContractOnly** switch.

|  |
| --- |
| **Description: Aa738737.note(en-us,VS.100).gifNote:** |
| Although ASP.NET Web services use the **XmlSerializer**, and WCF ASP.NET compatibility mode makes WCF services mimic the behavior of ASP.NET Web services, the ASP.NET compatibility option does not restrict one to using the **XmlSerializer**. One can still use the **DataContractSerializer** with services running in the ASP.NET compatibility mode. |

**Service Development**

To develop a service using ASP.NET, it has been customary to add the [WebService](http://msdn.microsoft.com/en-us/library/system.web.services.webservice.aspx) attribute to a class, and the [WebMethodAttribute](http://msdn.microsoft.com/en-us/library/system.web.services.webmethodattribute.aspx) to any of that class’ methods that are to be operations of the service:

[WebService]

public class Service : T:System.Web.Services.WebService

{

[WebMethod]

public string Echo(string input)

{

return input;

}

}

ASP.NET 2.0 introduced the option of adding the attribute **WebService** and **WebMethodAttribute** to an interface rather than to a class, and writing a class to implement the interface:

[WebService]

public interface IEcho

{

[WebMethod]

string Echo(string input);

}

public class Service : IEcho

{

public string Echo(string input)

{

return input;

}

}

Using this option is to be preferred, because the interface with the **WebService** attribute constitutes a contract for the operations performed by the service that can be reused with various classes that might implement that same contract in different ways.

A WCF service is provided by defining one or more WCF endpoints. An endpoint is defined by an address, a binding and a service contract. The address defines where the service is located. The binding specifies how to communicate with the service. The service contract defines the operations that the service can perform.

The service contract is usually defined first, by adding [ServiceContractAttribute](http://msdn.microsoft.com/en-us/library/system.servicemodel.servicecontractattribute.aspx) and [OperationContractAttribute](http://msdn.microsoft.com/en-us/library/system.servicemodel.operationcontractattribute.aspx) to an interface:

[ServiceContract]

public interface IEcho

{

[OperationContract]

string Echo(string input);

}

The **ServiceContractAttribute** specifies that the interface defines a WCF service contract, and the **OperationContractAttribute** indicates which, if any, of the methods of the interface define operations of the service contract.

Once a service contract has been defined, it is implemented in a class, by having the class implement the interface by which the service contract is defined:

public class Service : IEcho

{

public string Echo(string input)

{

return input;

}

}

A class that implements a service contract is referred to as a service type in WCF.

The next step is to associate an address and a binding with a service type. That is typically done in a configuration file, either by editing the file directly, or by using a configuration editor provided with WCF. Here is an example of a configuration file.

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<system.serviceModel>

<services>

<service name="Service ">

<endpoint

address="EchoService"

binding="basicHttpBinding"

contract="IEchoService "/>

</service>

</services>

</system.serviceModel>

</configuration>

The binding specifies the set of protocols for communicating with the application. The following table lists the system-provided bindings that represent common options.

|  |  |
| --- | --- |
| **Name** | **Purpose** |
| BasicHttpBinding | Interoperability with Web services and clients supporting the WS-BasicProfile 1.1 and Basic Security Profile 1.0. |
| WSHttpBinding | Interoperability with Web services and clients that support the WS-\* protocols over HTTP. |
| WSDualHttpBinding | Duplex HTTP communication, by which the receiver of an initial message does not reply directly to the initial sender, but may transmit any number of responses over a period of time by using HTTP in conformity with WS-\* protocols. |
| WSFederationBinding | HTTP communication, in which access to the resources of a service can be controlled based on credentials issued by an explicitly-identified credential provider. |
| NetTcpBinding | Secure, reliable, high-performance communication between WCF software entities across a network. |
| NetNamedPipeBinding | Secure, reliable, high-performance communication between WCF software entities on the same machine. |
| NetMsmqBinding | Communication between WCF software entities by using MSMQ. |
| MsmqIntegrationBinding | Communication between a WCF software entity and another software entity by using MSMQ. |
| NetPeerTcpBinding | Communication between WCF software entities by using Windows Peer-to-Peer Networking. |

The system-provided binding, [BasicHttpBinding](http://msdn.microsoft.com/en-us/library/system.servicemodel.basichttpbinding.aspx), incorporates the set of protocols supported by ASP.NET Web services.

Custom bindings for WCF applications are easily defined as collections of the binding element classes that WCF uses to implement individual protocols. New binding elements can be written to represent additional protocols.

The internal behavior of service types can be adjusted using the properties of a family of classes called behaviors. Here, the [ServiceBehaviorAttribute](http://msdn.microsoft.com/en-us/library/system.servicemodel.servicebehaviorattribute.aspx) class is used to specify that the service type is to be multithreaded.

[ServiceBehavior(ConcurrencyMode=ConcurrencyMode.Multiple]

public class DerivativesCalculatorServiceType: IDerivativesCalculator

Some behaviors, like **ServiceBehaviorAttribute**, are attributes. Others, the ones with properties that administrators would want to set, can be modified in the configuration of an application.

In programming service types, frequent use is made of the [OperationContext](http://msdn.microsoft.com/en-us/library/system.servicemodel.operationcontext.aspx) class. Its static [Current](http://msdn.microsoft.com/en-us/library/system.servicemodel.operationcontext.current.aspx) property provides access to information about the context in which an operation is running. **OperationContext** is similar to both the [HttpContext](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.aspx) and [ContextUtil](http://msdn.microsoft.com/en-us/library/system.enterpriseservices.contextutil.aspx) classes.

**Hosting**

ASP.NET Web services are compiled into a class library assembly. A file called the service file is provided that has the extension .asmx and contains an @ WebService directive that identifies the class that contains the code for the service and the assembly in which it is located.

<%@ WebService Language="C#" Class="Service,ServiceAssembly" %>

The service file is copied into an ASP.NET application root in Internet Information Services (IIS), and the assembly is copied into the \bin subdirectory of that application root. The application is then accessible by using the uniform resource locator (URL) of the service file in the application root.

WCF services can readily be hosted within IIS 5.1 or 6.0, the Windows Process Activation Service (WAS) that is provided as part of IIS 7.0, and within any .NET application. To host a service in IIS 5.1 or 6.0, the service must use HTTP as the communications transport protocol.

To host a service within IIS 5.1, 6.0 or within WAS, use the follows steps:

1. Compile the service type into a class library assembly.
2. Create a service file with a .svc extension with an @ ServiceHost directive to identify the service type:

<%@ServiceHost language=”c#” Service="MyService" %>

1. Copy the service file into a virtual directory, and the assembly into the \bin subdirectory of that virtual directory.
2. Copy the configuration file into the virtual directory, and name it Web.config.

The application is then accessible by using the URL of the service file in the application root.

To host a WCF service within a .NET application, compile the service type into a class library assembly referenced by the application, and program the application to host the service using the [ServiceHost](http://msdn.microsoft.com/en-us/library/system.servicemodel.servicehost.aspx) class. The following is an example of the basic programming required:

string httpBaseAddress = "http://www.contoso.com:8000/";

string tcpBaseAddress = "net.tcp://www.contoso.com:8080/";

Uri httpBaseAddressUri = new Uri(httpBaseAddress);

Uri tcpBaseAddressUri = new Uri(tcpBaseAddress);

Uri[] baseAdresses = new Uri[] {

httpBaseAddressUri,

tcpBaseAddressUri};

using(ServiceHost host = new ServiceHost(

typeof(Service), //”Service” is the name of the service type baseAdresses))

{

host.Open();

[…] //Wait to receive messages

host.Close();

}

This example shows how addresses for one or more transport protocols are specified in the construction of a **ServiceHost**. These addresses are referred to as base addresses.

The address provided for any endpoint of a WCF service is an address relative to a base address of the endpoint’s host. The host can have one base address for each communication transport protocol. In the sample configuration in the preceding configuration file, the **BasicHttpBinding** selected for the endpoint uses HTTP as the transport protocol, so the address of the endpoint, EchoService, is relative to the host’s HTTP base address. In the case of the host in the preceding example, the HTTP base address is http://www.contoso.com:8000/. For a service hosted within IIS or WAS, the base address is the URL of the service’s service file.

Only services hosted in IIS or WAS, and which are configured with HTTP as the transport protocol exclusively, can be made to use WCF ASP.NET compatibility mode option. Turning that option on requires the following steps.

1. The programmer must add the [AspNetCompatibilityRequirementsAttribute](http://msdn.microsoft.com/en-us/library/system.servicemodel.activation.aspnetcompatibilityrequirementsattribute.aspx) attribute to the service type and specify that ASP.NET compatibility mode is either allowed or required.

[System.ServiceModel.Activation.AspNetCompatibilityRequirements(

RequirementsMode=AspNetCompatbilityRequirementsMode.Require)]

public class DerivativesCalculatorServiceType: IDerivativesCalculator

1. The administrator must configure the application to use the ASP.NET compatibility mode.

<configuration>

<system.serviceModel>

<services>

[…]

</services>

<serviceHostingEnvironment aspNetCompatibilityEnabled="true"/>

</system.serviceModel>

</configuration>

WCF applications can also be configured to use .asmx as the extension for their service files rather than .svc.

<system.web>

<compilation>

<compilation debug="true">

<buildProviders>

<remove extension=".asmx"/>

<add extension=".asmx"

type="System.ServiceModel.ServiceBuildProvider,

Systemm.ServiceModel,

Version=3.0.0.0,

Culture=neutral,

PublicKeyToken=b77a5c561934e089" />

</buildProviders>

</compilation>

</compilation>

</system.web>

That option can save you from having to modify clients that are configured to use the URLs of .asmx service files when modifying a service to use WCF.

**Client Development**

Clients for ASP.NET Web services are generated using the command-line tool, WSDL.exe, which provides the URL of the .asmx file as input. The corresponding tool provided by WCF is [ServiceModel Metadata Utility Tool (Svcutil.exe)](http://msdn.microsoft.com/en-us/library/aa347733.aspx). It generates a code module with the definition of the service contract and the definition of a WCF client class. It also generates a configuration file with the address and binding of the service.

In programming a client of a remote service it is generally advisable to program according to an asynchronous pattern. The code generated by the WSDL.exe tool always provides for both a synchronous and an asynchronous pattern by default. The code generated by the [ServiceModel Metadata Utility Tool (Svcutil.exe)](http://msdn.microsoft.com/en-us/library/aa347733.aspx) can provide for either pattern. It provides for the synchronous pattern by default. If the tool is executed with the **/async** switch, then the generated code provides for the asynchronous pattern.

There is no guarantee that names in the WCF client classes generated by ASP.NET’s WSDL.exe tool, by default, match the names in WCF client classes generated by the Svcutil.exe tool. In particular, the names of the properties of classes that have to be serialized using the **XmlSerializer** are, by default, given the suffix Property in the code generated by the Svcutil.exe tool, which is not the case with the WSDL.exe tool.

**Message Representation**

The headers of the SOAP messages sent and received by ASP.NET Web services can be customized. A class is derived from [SoapHeader](http://msdn.microsoft.com/en-us/library/system.web.services.protocols.soapheader.aspx) to define the structure of the header, and then the [SoapHeaderAttribute](http://msdn.microsoft.com/en-us/library/system.web.services.protocols.soapheaderattribute.aspx) is used to indicate the presence of the header.

public class SomeProtocol : SoapHeader

{

public long CurrentValue;

public long Total;

}

[WebService]

public interface IEcho

{

SomeProtocol ProtocolHeader

{

get;

set;

}

[WebMethod]

[SoapHeader("ProtocolHeader")]

string PlaceOrders(PurchaseOrderType order);

}

public class Service: WebService, IEcho

{

private SomeProtocol protocolHeader;

public SomeProtocol ProtocolHeader

{

get

{

return this.protocolHeader;

}

set

{

this.protocolHeader = value;

}

}

string PlaceOrders(PurchaseOrderType order)

{

long currentValue = this.protocolHeader.CurrentValue;

}

}

The WCF provides the attributes, [MessageContractAttribute](http://msdn.microsoft.com/en-us/library/system.servicemodel.messagecontractattribute.aspx), [MessageHeaderAttribute](http://msdn.microsoft.com/en-us/library/system.servicemodel.messageheaderattribute.aspx), and [MessageBodyMemberAttribute](http://msdn.microsoft.com/en-us/library/system.servicemodel.messagebodymemberattribute.aspx) to describe the structure of the SOAP messages sent and received by a service.

[DataContract]

public class SomeProtocol

{

[DataMember]

public long CurrentValue;

[DataMember]

public long Total;

}

[DataContract]

public class Item

{

[DataMember]

public string ItemNumber;

[DataMember]

public decimal Quantity;

[DataMember]

public decimal UnitPrice;

}

[MessageContract]

public class ItemMesage

{

[MessageHeader]

public SomeProtocol ProtocolHeader;

[MessageBody]

public Item Content;

}

[ServiceContract]

public interface IItemService

{

[OperationContract]

public void DeliverItem(ItemMessage itemMessage);

}

This syntax yields an explicit representation of the structure of the messages, whereas the structure of messages is implied by the code of an ASP.NET Web service. Also, in the ASP.NET syntax, message headers are represented as properties of the service, such as the ProtocolHeader property in the previous example, whereas in WCF syntax, they are more accurately represented as properties of messages. Also, WCF allows message headers to be added to the configuration of endpoints.

<service name="Service ">

<endpoint

address="EchoService"

binding="basicHttpBinding"

contract="IEchoService ">

<headers>

<dsig:X509Certificate

xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">

...

</dsig:X509Certificate>

</headers>

</endpoint>

</service>

That option allows you to avoid any reference to infrastructural protocol headers in the code for a client or service: the headers are added to messages because of how the endpoint is configured.

**Service Description**

Issuing an HTTP GET request for the .asmx file of an ASP.NET Web service with the query WSDL causes ASP.NET to generate WSDL to describe the service. It returns that WSDL as the response to the request.

ASP.NET 2.0 made it possible to validate that a service is compliant with the Basic Profile 1.1 of the Web Services-Interoperability Organization (WS-I), and to insert a claim that the service is compliant into its WSDL. That is done using the ConformsTo and EmitConformanceClaims parameters of the [WebServiceBindingAttribute](http://msdn.microsoft.com/en-us/library/system.web.services.webservicebindingattribute.aspx) attibute.

[WebService(Namespace = "http://tempuri.org/")]

[WebServiceBinding(

ConformsTo = WsiProfiles.BasicProfile1\_1,

EmitConformanceClaims=true)]

public interface IEcho

The WSDL that ASP.NET generates for a service can be customized. Customizations are made by creating a derived class of [ServiceDescriptionFormatExtension](http://msdn.microsoft.com/en-us/library/system.web.services.description.servicedescriptionformatextension.aspx) to add items to the WSDL.

Issuing an HTTP GET request with the query WSDL for the .svc file of a WCF service with an HTTP endpoint hosted within IIS 51, 6.0 or WAS causes WCF to respond with WSDL to describe the service. Issuing an HTTP GET request with the query WSDL to the HTTP base address of a service hosted within a .NET application has the same effect if httpGetEnabled is set to true.

However, WCF also responds to WS-MetadataExchange requests with WSDL that it generates to describe a service. ASP.NET Web services do not have built-in support for WS-MetadataExchange requests.

The WSDL that WCF generates can be extensively customized. The [ServiceMetadataBehavior](http://msdn.microsoft.com/en-us/library/system.servicemodel.description.servicemetadatabehavior.aspx) class provides some facilities for customizing the WSDL. The WCF can also be configured to not generate WSDL, but rather to use a static WSDL file at a given URL.

<behaviors>

<behavior name="DescriptionBehavior">

<metadataPublishing

enableMetadataExchange="true"

enableGetWsdl="true"

enableHelpPage="true"

metadataLocation=

"http://localhost/DerivativesCalculatorService/Service.WSDL"/>

</behavior>

</behaviors>

**Exception Handling**

In ASP.NET Web services, unhandled exceptions are returned to clients as SOAP faults. You can also explicitly throw instances of the [SoapException](http://msdn.microsoft.com/en-us/library/system.web.services.protocols.soapexception.aspx) class and have more control over the content of the SOAP fault that gets transmitted to the client.

In WCF services, unhandled exceptions are not returned to clients as SOAP faults to prevent sensitive information being inadvertently exposed through the exceptions. A configuration setting is provided to have unhandled exceptions returned to clients for the purpose of debugging.

To return SOAP faults to clients, you can throw instances of the generic type, [FaultException](http://msdn.microsoft.com/en-us/library/ms576199.aspx), using the data contract type as the generic type. You can also add [FaultContractAttribute](http://msdn.microsoft.com/en-us/library/system.servicemodel.faultcontractattribute.aspx) attributes to operations to specify the faults that an operation might yield.

[DataContract]

public class MathFault

{

[DataMember]

public string operation;

[DataMember]

public string problemType;

}

[ServiceContract]

public interface ICalculator

{

[OperationContract]

[FaultContract(typeof(MathFault))]

int Divide(int n1, int n2);

}

Doing so results in the possible faults being advertised in the WSDL for the service, allowing client programmers to anticipate which faults can result from an operation, and write the appropriate catch statements.

try

{

result = client.Divide(value1, value2);

}

catch (FaultException<MathFault> e)

{

Console.WriteLine("FaultException<MathFault>: Math fault while doing "

+ e.Detail.operation

+ ". Problem: "

+ e.Detail.problemType);

}

**State Management**

The class used to implement an ASP.NET Web service may be derived from **WebService**.

public class Service : WebService, IEcho

{

public string Echo(string input)

{

return input;

}

}

In that case, the class can be programmed to use the **WebService** base class’ Context property to access a **HttpContext** object. The **HttpContext** object can be used to update and retrieve application state information by using its Application property, and can be used to update and retrieve session state information by using its Session property.

ASP.NET provides considerable control over where the session state information accessed by using the Session property of the **HttpContext** is actually stored. It may be stored in cookies, in a database, in the memory of the current server, or in the memory of a designated server. The choice is made in the service’s configuration file.

The WCF provides extensible objects for state management. Extensible objects are objects that implement [IExtensibleObject](http://msdn.microsoft.com/en-us/library/ms586697.aspx). The most important extensible objects are [ServiceHostBase](http://msdn.microsoft.com/en-us/library/system.servicemodel.servicehostbase.aspx) and [InstanceContext](http://msdn.microsoft.com/en-us/library/system.servicemodel.instancecontext.aspx). **ServiceHostBase** allows you to maintain state that all of the instances of all of the service types on the same host can access, while **InstanceContext** allows you to maintain state that can be accessed by any code running within the same instance of a service type.

Here, the service type, TradingSystem, has a **ServiceBehaviorAttribute** that specifies that all calls from the same WCF client instance are routed to the same instance of the service type.

[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)]

public class TradingSystem: ITradingService

The class, DealData, defines state that can be accessed by any code running in the same instance of a service type.

internal class DealData: IExtension<InstanceContext>

{

public string DealIdentifier = null;

public Trade[] Trades = null;

}

In the code of the service type that implements one of the operations of the service contract, a DealData state object is added to the state of the current instance of the service type.

string ITradingService.BeginDeal()

{

string dealIdentifier = Guid.NewGuid().ToString();

DealData state = new DealData(dealIdentifier);

OperationContext.Current.InstanceContext.Extensions.Add(state);

return dealIdentifier;

}

That state object can then be retrieved and modified by the code that implements another of the service contract’s operations.

void ITradingService.AddTrade(Trade trade)

{

DealData dealData = OperationContext.Current.InstanceContext.Extensions.Find<DealData>();

dealData.AddTrade(trade);

}

Whereas ASP.NET provides control over where state information in the **HttpContext** class is actually stored, WCF, at least in its initial version, provides no control over where extensible objects are stored. That constitutes the very best reason for selecting the ASP.NET compatibility mode for a WCF service. If configurable state management is imperative, then opting for the ASP.NET compatibility mode allows you to use the facilities of the **HttpContext** class exactly as they are used in ASP.NET, and also to configure where state information managed by using the **HttpContext** class is stored.

**Security**

The options for securing ASP.NET Web services are those for securing any IIS application. Because WCF applications can be hosted not only within IIS but also within any .NET executable, the options for securing WCF applications must be made independent from the facilities of IIS. However, the facilities provided for ASP.NET Web services are also available for WCF services running in ASP.NET compatibility mode.

**Security: Authentication**

IIS provides facilities for controlling access to applications by which you can select either anonymous access or a variety of modes of authentication: Windows Authentication, Digest Authentication, Basic Authentication, and .NET Passport Authentication. The Windows Authentication option can be used to control access to ASP.NET Web services. However, when WCF applications are hosted within IIS, IIS must be configured to permit anonymous access to the application, so that authentication can be managed by WCF itself, which does support Windows authentication among various other options. The other options that are built-in include username tokens, X.509 certificates, SAML tokens, and CardSpace card, but custom authentication mechanisms can also be defined.

**Security: Impersonation**

ASP.NET provides an identity element by which an ASP.NET Web service can be made to impersonate a particular user or whichever user’s credentials are provided with the current request. That element can be used to configure impersonation in WCF applications running in ASP.NET compatibility mode.

The WCF configuration system provides its own identity element for designating a particular user to impersonate. Also, WCF clients and services can be independently configured for impersonation. Clients can be configured to impersonate the current user when they transmit requests.

<behaviors>

<behavior name="DerivativesCalculatorClientBehavior">

<clientCredentials>

<windows allowedImpersonationLevel="Impersonation"/>

</clientCredentials>

</behavior>

</behaviors>

Service operations can be configured to impersonate whichever user’s credentials are provided with the current request.

[OperationBehavior(Impersonation = ImpersonationOption.Required)]

public void Receive(Message input)

**Security: Authorization using Access Control Lists**

Access Control Lists (ACLs) can be used to restrict access to .asmx files. However, ACLs on WCF .svc files are ignored except in ASP.NET compatibility mode.

**Security: Role-based Authorization**

The IIS Windows Authentication option can be used in conjunction with the authorization element provided by the ASP.NET configuration language to facilitate role-based authorization for ASP.NET Web services based on the Windows groups to which users are assigned. ASP.NET 2.0 introduced a more general role-based authorization mechanism: role providers.

Role providers are classes that all implement a basic interface for enquiring about the roles to which a user is assigned, but each role provider knows how to retrieve that information from a different source. ASP.NET 2.0 provides a role provider that can retrieve role assignments from a Microsoft SQL Server database, and another that can retrieve role assignments from the Windows Server 2003 Authorization Manager.

The role provider mechanism can actually be used independently of ASP.NET in any .NET application, including a WCF application. The following sample configuration for a WCF application shows how the use of an ASP.NET role provider is an option selected by means of the [ServiceAuthorizationBehavior](http://msdn.microsoft.com/en-us/library/system.servicemodel.description.serviceauthorizationbehavior.aspx).

<system.serviceModel>

<services>

<service name="Service.ResourceAccessServiceType"

behaviorConfiguration="ServiceBehavior">

<endpoint

address="ResourceAccessService"

binding="wsHttpBinding"

contract="Service.IResourceAccessContract"/>

</service>

</services>

<behaviors>

<behavior name="ServiceBehavior">

<serviceAuthorization principalPermissionMode="UseAspNetRoles"/>

</behavior>

</behaviors>

</system.serviceModel>

**Security: Claims-based Authorization**

One of the most important innovations of WCF is its thorough support for authorizing access to protected resources based on claims. Claims consist of a type, a right and a value, a drivers’ license, for example. It makes a set of claims about the bearer, one of which is the bearer’s date of birth. The type of that claim is date of birth, while the value of the claim is the driver’s birth date. The right that a claim confers on the bearer specifies what the bearer can do with the claim’s value. In the case of the claim of the driver’s date of birth, the right is possession: the driver possesses that date of birth but cannot, for example, alter it. Claims-based authorization encloses role-based authorization, because roles are a type of claim.

Authorization based on claims is accomplished by comparing a set of claims to the access requirements of the operation and, depending on the outcome of that comparison, granting or denying access to the operation. In WCF, you can specify a class to use to run claims-based authorization, once again by assigning a value to the ServiceAuthorizationManager property of **ServiceAuthorizationBehavior**.

<behaviors>

<behavior name='ServiceBehavior'>

<serviceAuthorization

serviceAuthorizationManagerType=

'Service.AccessChecker, Service' />

</behavior>

</behaviors>

Classes used to run claims-based authorization must derive from [ServiceAuthorizationManager](http://msdn.microsoft.com/en-us/library/system.servicemodel.serviceauthorizationmanager.aspx), which has just one method to override, AccessCheck(). WCF calls that method whenever an operation of the service is invoked and provides a **OperationContext** object, which has the claims for the user in its ServiceSecurityContext.AuthorizationContext property. WCF does the work of assembling claims about the user from whatever security token the user provided for authentication, which leaves the of task of evaluating whether those claims suffice for the operation in question. That WCF automatically assembles claims from any kind of security token is a highly significant innovation, because it makes the code for authorization based on the claims entirely independent of the authentication mechanism. By contrast, authorization using ACLs or roles in ASP.NET is closely tied to Windows authentication.

**Security: Confidentiality** The confidentiality of messages exchanged with ASP.NET Web services can be ensured at the transport level by configuring the application within IIS to use the Secure Hypertext Transfer Protocol (HTTPS). The same can be done for WCF applications hosted within IIS. However, WCF applications hosted outside of IIS can also be configured to use a secure transport protocol. More important, WCF applications can also be configured to secure the messages before they are transported, using the WS-Security protocol. Securing just the body of a message using WS-Security allows it to be transmitted confidentially across intermediaries before reaching its final destination.

**Globalization**  The ASP.NET configuration language allows you to specify the culture for individual services. The WCF does not support that configuration setting except in ASP.NET compatibility mode. To localize a WCF service that does not use ASP.NET compatibility mode, compile the service type into culture-specific assemblies, and have separate culture-specific endpoints for each culture-specific assembly.