

Contents

Service-Oriented Applications with WCF

[What's New in Windows Communication Foundation 4.5](#)

[WCF Simplification Features](#)

[Guide to the Documentation](#)

[Conceptual Overview](#)

[What Is Windows Communication Foundation](#)

[Fundamental Windows Communication Foundation Concepts](#)

[Windows Communication Foundation Architecture](#)

[WCF and .NET Framework Client Profile](#)

[Getting Started Tutorial](#)

[Define a Service Contract](#)

[Implement a Service Contract](#)

[Host and Run a Basic Service](#)

[Create a Client](#)

[Configure a Client](#)

[Use a Client](#)

[Troubleshoot the Getting Started Tutorial](#)

[Basic WCF Programming](#)

[Basic Programming Lifecycle](#)

[Designing and Implementing Services](#)

[Designing Service Contracts](#)

[Specifying and Handling Faults in Contracts and Services](#)

[Defining and Specifying Faults](#)

[Sending and Receiving Faults](#)

[Using Sessions](#)

[Synchronous and Asynchronous Operations](#)

[How to: Implement an Asynchronous Service Operation](#)

[Reliable Services](#)

[Services and Transactions](#)

Implementing Service Contracts

Specifying Service Run-Time Behavior

Configuring Services

Simplified Configuration

Configuring Services Using Configuration Files

Configuring WCF Services in Code

Bindings

WCF Bindings Overview

System-Provided Bindings

Using Bindings to Configure Services and Clients

How to: Specify a Client Binding in Code

How to: Specify a Client Binding in Configuration

How to: Specify a Service Binding in Code

How to: Specify a Service Binding in Configuration

Configuring Bindings for Services

Endpoints

Endpoint Creation Overview

Specifying an Endpoint Address

Publishing Metadata Endpoints

Securing Services

How to: Secure a Service with Windows Credentials

How to: Set the Security Mode

How to: Specify the Client Credential Type

How to: Restrict Access with the `PrincipalPermissionAttribute` Class

How to: Impersonate a Client on a Service

How to: Examine the Security Context

Understanding Protection Level

How to: Set the `ProtectionLevel` Property

Creating WS-I Basic Profile 1.1 Interoperable Services

Hosting Services

How to: Host a WCF Service in a Managed Application

How to: Host a WCF Service Written with .NET Framework 3.5 in IIS Running Under .NET Framework 4

Building Clients

- WCF Client Overview

- Accessing Services Using a WCF Client

 - Add Service Reference in a Portable Subset Project

 - Specifying Client Run-Time Behavior

 - Configuring Client Behaviors

- Securing Clients

 - How to: Specify Client Credential Values

- Introduction to Extensibility

- WCF Troubleshooting Quickstart

- WCF Error Handling

- WCF and ASP.NET Web API

WCF Feature Details

- Extending WCF

- Guidelines and Best Practices

 - Best Practices: Data Contract Versioning

 - Best Practices: Intermediaries

 - Service Versioning

 - Load Balancing

 - Controlling Resource Consumption and Improving Performance

 - Deploying WCF Applications with ClickOnce

- Administration and Diagnostics

- System Requirements

- Operating System Resources Required by WCF

- Troubleshooting Setup Issues

- Migrating from .NET Remoting to WCF

- Using the WCF Development Tools

 - WCF Service Host (WcfSvcHost.exe)

 - WCF Test Client (WcfTestClient.exe)

 - WCF Visual Studio Templates

 - WCF Service Publishing

 - Renaming a WCF Service

[Deploying a WCF Library Project](#)

[Controlling Auto-launching of WCF Service Host](#)

[Generating Data Type Classes from XML](#)

[Windows Communication Foundation Tools](#)

[ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#)

[Find Private Key Tool \(FindPrivateKey.exe\)](#)

[ServiceModel Registration Tool \(ServiceModelReg.exe\)](#)

[Service Trace Viewer Tool \(SvcTraceViewer.exe\)](#)

[Configuration Editor Tool \(SvcConfigEditor.exe\)](#)

[COM+ Service Model Configuration Tool \(ComSvcConfig.exe\)](#)

[WS-AtomicTransaction Configuration Utility \(wsatConfig.exe\)](#)

[Interpreting Error Codes Returned by wsatConfig.exe](#)

[WS-AtomicTransaction Configuration MMC Snap-in](#)

[WorkFlow Service Registration Tool \(WFServicesReg.exe\)](#)

[Contract-First Tool](#)

[Windows Communication Foundation Samples](#)

[Windows Communication Foundation Glossary](#)

[General Reference](#)

[Feedback and Community](#)

[Privacy Information](#)

Developing Service-Oriented Applications with WCF

5/5/2018 • 2 minutes to read • [Edit Online](#)

This section of the documentation provides information about Windows Communication Foundation (WCF), which is a unified programming model for building service-oriented applications. It enables developers to build secure, reliable, transacted solutions that integrate across platforms and interoperate with existing investments.

In this section

[What's New in Windows Communication Foundation 4.5](#)

Discusses features new to Windows Communication Foundation.

[WCF Simplification Features](#)

Discusses new features that make writing WCF applications simpler.

[Guide to the Documentation](#)

A description of the WCF documentation

[Conceptual Overview](#)

Summarizes information about the Windows Communication Foundation (WCF) messaging system and the classes that support its use.

[Getting Started Tutorial](#)

A step by step tutorial to create a WCF service and client

[Basic WCF Programming](#)

Describes the fundamentals for creating Windows Communication Foundation applications.

[WCF Feature Details](#)

Shows topics that let you choose which WCF feature or features you need to employ.

[Extending WCF](#)

Describes how to modify and extend WCF runtime components

[Guidelines and Best Practices](#)

Provides guidelines for creating Windows Communication Foundation (WCF) applications.

[Administration and Diagnostics](#)

Describes the diagnostic features of WCF

[System Requirements](#)

Describes system requirements needed to run WCF

[Operating System Resources Required by WCF](#)

Describes operating system resources required by WCF

[Troubleshooting Setup Issues](#)

Provides guidance for fixing WCF setup issues

[Migrating from .NET Remoting to WCF](#)

Compares .NET Remoting to WCF and provides migration guidance for common scenarios.

[Using the WCF Development Tools](#)

Describes the Visual Studio Windows Communication Foundation development tools that can assist you in developing your WCFservice.

[Windows Communication Foundation Tools](#)

Describes WCF tools designed to make it easier to create, deploy, and manage WCF applications

[Windows Communication Foundation Samples](#)

Samples that provide instruction on various aspects of Windows Communication Foundation

[Windows Communication Foundation Glossary](#)

Shows a list of terms specific to WCF

[General Reference](#)

The section describes the elements that are used to configure Windows Communication Foundation clients and services.

[Feedback and Community](#)

Information about how to provide feedback about Windows Communication Foundation

[Privacy Information](#)

Information regarding WCF and Privacy

What's New in Windows Communication Foundation 4.5

10/4/2018 • 8 minutes to read • [Edit Online](#)

This topic discusses features new to Windows Communication Foundation (WCF) version 4.5.

WCF Simplification Features

Much work has been done to make WCF 4.5 applications easier to develop and maintain. For more information, see [WCF Simplification Features](#).

Task-based Async Support

By default, Add Service Reference generates Task-returning async service operation methods. This is done for both synchronous and asynchronous methods. This allows you to call the service operations asynchronously using the new Task based async programming model. When you call the generated proxy method, WCF constructs a Task object to represent the asynchronous operation and returns that Task to you. The Task completes when the operation completes. When implementing an async operation you can implement it as a task-based async operation. For more information see, [Synchronous and Asynchronous Operations](#).

Simplified Generated Configuration Files

When you add a service reference in Visual Studio or use the SvcUtil.exe tool, a client configuration file is generated. In previous versions of WCF these configuration files contained the value of every binding property even if its value is the default value. In WCF 4.5 the generated configuration files contain only those binding properties that are set to a non-default value.

For more information, see [WCF Simplification Features](#)

Contract-First Development

WCF now has support for contract-first development. The svcutil.exe has a /serviceContract switch which allows you to generate service and data contracts from a WSDL document.

Add Service Reference from a Portable Subset Project

Portable subset projects enable .NET assembly programmers to maintain a single source tree and build system while still supporting multiple .NET platforms (desktop, Silverlight, Windows Phone, and XBOX). Portable subset projects only reference .NET portable libraries which are a .NET framework assembly that can be used on any .NET platform. The developer experience is the same as adding a service reference within any other WCF client application. For more information, see [Add Service Reference in a Portable Subset Project](#).

ASP.NET Compatibility Mode Default Changed

WCF provides ASP.NET compatibility mode to grant developers full access to the features in the ASP.NET HTTP pipeline when writing WCF services. To use this mode, you must set the `aspNetCompatibilityEnabled` attribute to true in the `<serviceHostingEnvironment>` section of web.config. Additionally, any service in this appDomain needs to have the `RequirementsMode` property on its `AspNetCompatibilityRequirementsAttribute` set to `Allowed` or `Required`. By default `AspNetCompatibilityRequirementsAttribute` is now set to `Allowed`. For more information, see [What's New in Windows Communication Foundation](#) and [WCF Services and ASP.NET](#).

New Transport Default Values

In order to simplify configuration a number of transport property default values have changed. For more information, see [WCF Simplification Features](#).

XmlDictionaryReaderQuotas

[XmlDictionaryReaderQuotas](#) contains configurable quota values for XML dictionary readers which limit the amount of memory utilized by an encoder while creating a message. While these quotas are configurable, the default values have changed to lessen the possibility that a developer will have to set them explicitly. For more information, see [WCF Simplification Features](#).

WCF Configuration Validation

As part of the build process within Visual Studio, WCF configuration files are now validated for attributes defined within the project. A list of validation errors or warnings is displayed in Visual Studio if the validation fails.

XML Editor Tooltips

In order to help new and existing WCF service developers to configure their services, the Visual Studio XML editor now provides tooltips for every configuration element and its properties that is part of the service configuration file.

Streaming Improvements

Added support for true asynchronous streaming where the send side now does not block threads if the receive side is not reading or slow in reading thereby increasing scalability. Removed the limitation of message buffering when a client sends a streamed message to an IIS hosted WCF service. For more information, see [WCF Simplification Features](#).

Simplifying Exposing an Endpoint Over HTTPS with IIS

An HTTPS protocol mapping has been added to simplify exposing an endpoint over HTTPS. To enable an HTTPS endpoint, ensure your website has an HTTPS binding and SSL certificate configured, and then simply enable HTTPS for the virtual directory that hosts the service. If metadata is enabled for the service, it will be exposed over HTTPS as well.

Generating a Single WSDL Document

Some third-party WSDL processing stacks are not able to process WSDL documents that have dependencies on other documents through a `xsd:import`. WCF now allows you to specify that all WSDL information be returned in a single document. To request a single WSDL document append "?singleWSDL" to the URI when requesting metadata from the service.

WebSocket Support

WebSockets is a technology that provides true bidirectional communication over ports 80 and 443 with performance characteristics similar to TCP. Two new bindings have been added to support communication over a WebSocket transport. [NetHttpBinding](#) and [NetHttpsBinding](#). For more information see: [System-Provided Bindings](#).

New Transport Default Values

The following table describes the settings that have changed and where to find additional information.

PROPERTY	ON	NEW DEFAULT	FOR MORE INFORMATION SEE
channelInitializationTimeout	NetTcpBinding	30 seconds	ChannelInitializationTimeout
listenBacklog	NetTcpBinding	12 * number of processors	ListenBacklog

PROPERTY	ON	NEW DEFAULT	FOR MORE INFORMATION SEE
maxPendingAccepts	ConnectionOrientedTransportBindingElement SMSSvcHost.exe	2 * number of processors for transport 4 * number of processors for SMSSvcHost.exe	MaxPendingAccepts Configuring the Net.TCP Port Sharing Service
maxPendingConnections	ConnectionOrientedTransportBindingElement	12 * number of processors	MaxPendingConnections
receiveTimeout	SMSSvcHost.exe	30 seconds	Configuring the Net.TCP Port Sharing Service

XML Editor Tooltips

In order to help new and existing WCF service developers to configure their services, the Visual Studio XML editor now provides tooltips for every configuration element and its properties that is part of the service configuration file.

Configuring WCF Services in Code

Windows Communication Foundation (WCF) allows developers to configure services using configuration files or code. Configuration files are useful when a service needs to be configured after being deployed. When using configuration files, an IT professional only needs to update the configuration file, no recompilation is required. Configuration files, however, can be complex and difficult to maintain. There is no support for debugging configuration files and configuration elements are referenced by names which makes authoring configuration files error-prone and difficult. WCF also allows you to configure services in code. In earlier versions of WCF (4.0 and earlier) configuring services in code was easy in self-hosted scenarios, the [ServiceHost](#) class allowed you to configure endpoints and behaviors prior to calling `ServiceHost.Open`. In web hosted scenarios, however, you don't have access to the [ServiceHost](#) class. To configure a web hosted service you were required to create a `System.ServiceModel.ServiceHostFactory` that created the [ServiceHostFactory](#) and performed any needed configuration. Starting with .NET 4.5, WCF provides an easier way to configure both self-hosted and web hosted services in code. For more information, see [Configuring WCF Services in Code](#).

ChannelFactory Caching

WCF client applications use the [ChannelFactory<TChannel>](#) class to create a communication channel with a WCF service. Creating [ChannelFactory<TChannel>](#) instances incurs some overhead because it involves the following operations:

1. Constructing the [ContractDescription](#) tree
2. Reflecting all of the required CLR types
3. Constructing the channel stack
4. Disposing of resources

To help minimize this overhead, WCF can cache channel factories when you are using a WCF client proxy. For more information, see [Channel Factory and Caching](#).

Compression and the Binary Encoder

Beginning with WCF 4.5 the WCF binary encoder adds support for compression. The type of compression is

configured with the [CompressionFormat](#) property. Both the client and the service must configure the [CompressionFormat](#) property. Compression will work for HTTP, HTTPS, and TCP protocols. If a client specifies to use compression but the service does not support it a protocol exception is thrown indicating a protocol mismatch. For more information, see [Choosing a Message Encoder](#)

UDP

Support has been added for a UDP transport which allows developers to write services that use "fire and forget" messaging. A client sends a message to a service and expects no response from the service.

Multiple Authentication Support

Support has been added to support multiple authentication modes, as supported by IIS, on a single WCF endpoint when using the HTTP transport and transport security. IIS allows you to enable multiple authentication modes on a virtual directory, this feature allows a single WCF endpoint to support the multiple authentication modes enabled for the virtual directory where the WCF service is hosted.

IDN Support

Support has been added to allow for WCF services with Internationalized Domain Names. For more information see [WCF and Internationalized Domain Names](#).

HttpClient

A new class called [HttpClient](#) has been added to make working with HTTP requests much easier. For more info, see [Making apps social and connected with HTTP services](#) and the [HTTP Client Sample](#).

Configuration Intellisense

Attribute values in configuration files for custom attributes defined in the project now support intellisense to facilitate working with configurations quickly and accurately.

Configuration tooltips

WCF elements and attributes now have tooltips in the XML editor, to more easily and accurately identify the purpose of the element or attribute.

Paste Data as Classes

In a WCF project, data types defined in XML (such as are exposed in a service) can be pasted directly into a code page. The XML type will be pasted as a CLR type. See [Generating Data Type Classes from XML](#) for more details.

WebServiceHost and default endpoints

In Visual Studio 2010, WebServiceHost automatically created a default endpoint whether you explicitly specified an endpoint or not. In Visual Studio 2012 and later, WebServiceHost only creates a default endpoint if no endpoints are explicitly added. If your client is expecting the default endpoint you can explicitly add an endpoint and point the client to it. Alternatively, you can tell WCF to revert back to the previous behavior by adding the following setting to your application's configuration file

```
<appSettings>
  <add key="wcf:webservicehost:enableautomaticendpointscmpatability" value="true"/>
</appSettings>
```

IHttpClientContainerManager

This interface, exposed by [IChannelFactory<TChannel>](#), makes working with cookies on the client side much easier. When AllowCookies is set to true on the binding, you can access cookies by using the following code:

```
IHttpClientContainerManager cookieManager = factory.GetProperty<IHttpClientContainerManager>();  
System.Net.CookieContainer container = cookieManager.CookieContainer;
```

You can then retrieve or set the cookies from the [CookieContainer](#). When AllowCookies is set to false, you can manually retrieve the cookies using [OperationContext](#) and send it in other requests using another [OperationContext](#) or message inspector. The IHttpClientContainerManager interface allows you to authenticate a user with a service and use the authentication cookie returned by that service to authenticate with other services.

WCF Simplification Features

10/4/2018 • 7 minutes to read • [Edit Online](#)

This topic discusses new features that make writing WCF applications simpler.

Simplified Generated Configuration Files

When you add a service reference in Visual Studio or use the SvcUtil.exe tool a client configuration file is generated. In previous versions of WCF these configuration files contained the value of every binding property even if its value is the default value. In WCF 4.5 the generated configuration files contain only those binding properties that are set to a non-default value.

The following is an example of a configuration file generated by WCF 3.0.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="BasicHttpBinding_IService1" closeTimeout="00:01:00"
          openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
          allowCookies="false" bypassProxyOnLocal="false"
          hostNameComparisonMode="StrongWildcard" maxBufferSize="65536"
          maxBufferPoolSize="524288" maxReceivedMessageSize="65536"
          messageEncoding="Text" textEncoding="utf-8" transferMode="Buffered"
          useDefaultWebProxy="true">
          <readerQuotas maxDepth="32" maxStringContentLength="8192"
            maxArrayLength="16384" maxBytesPerRead="4096"
            maxNameTableCharCount="16384" />
          <security mode="None">
            <transport clientCredentialType="None" proxyCredentialType="None"
              realm="" />
            <message clientCredentialType="UserName" algorithmSuite="Default" />
          </security>
        </binding>
      </basicHttpBinding>
    </bindings>
    <client>
      <endpoint address="http://localhost:36906/Service1.svc" binding="basicHttpBinding"
        bindingConfiguration="BasicHttpBinding_IService1" contract="IService1"
        name="BasicHttpBinding_IService1" />
    </client>
  </system.serviceModel>
</configuration>
```

The following is an example of the same configuration file generated by WCF 4.5.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="BasicHttpBinding_IService1" />
      </basicHttpBinding>
    </bindings>
    <client>
      <endpoint address="http://localhost:36906/Service1.svc" binding="basicHttpBinding"
        bindingConfiguration="BasicHttpBinding_IService1" contract="IService1"
        name="BasicHttpBinding_IService1" />
    </client>
  </system.serviceModel>
</configuration>
```

Contract-First Development

WCF now has support for contract-first development. The svcutil.exe tool has a /serviceContract switch which allows you to generate service and data contracts from a WSDL document.

Add Service Reference From a Portable Subset Project

Portable subset projects enable .NET assembly programmers to maintain a single source tree and build system while still supporting multiple .NET implementations (desktop, Silverlight, Windows Phone, and XBOX). Portable subset projects only reference .NET portable libraries which are a .NET framework assembly that can be used on any .NET implementation. The developer experience is the same as adding a service reference within any other WCF client application. For more information, see [Add Service Reference in a Portable Subset Project](#).

ASP.NET Compatibility Mode Default Changed

WCF provides ASP.NET compatibility mode to grant developers full access to the features in the ASP.NET HTTP pipeline when writing WCF services. To use this mode, you must set the `aspNetCompatibilityEnabled` attribute to true in the `<serviceHostingEnvironment>` section of web.config. Additionally, any service in this appDomain needs to have the `RequirementsMode` property on its `AspNetCompatibilityRequirementsAttribute` set to `Allowed` or `Required`. By default `AspNetCompatibilityRequirementsAttribute` is now set to `Allowed` and the default WCF service application template sets the `aspNetCompatibilityEnabled` attribute to `true`. For more information, see [What's New in Windows Communication Foundation 4.5](#) and [WCF Services and ASP.NET](#).

Streaming Improvements

- New support for asynchronous streaming has been added to WCF. To enable asynchronous streaming, add the `DispatcherSynchronizationBehavior` endpoint behavior to the service host and set its `AsynchronousSendEnabled` property to `true`. This can benefit scalability when a service is sending streamed messages to multiple clients which are reading slowly. WCF does not block one thread per client anymore and will free up the thread to service another client.
- Removed limitations around buffering of messages when a service is IIS hosted. In previous versions of WCF when receiving a message for an IIS-hosted service that used streaming message transfer, ASP.NET would buffer the entire message before sending it to WCF. This would cause large memory consumption. This buffering has been removed in .NET 4.5 and now IIS-hosted WCF services can start processing the incoming stream before the entire message has been received, thereby enabling true streaming. This allows WCF to respond immediately to messages and allows improved performance. In addition, you no longer have to specify a value for `maxRequestLength`, the ASP.NET size limit on incoming requests. If this property is set, it is ignored. For more information about `maxRequestLength` see [<httpRuntime> configuration element](#). You will still need to configure the `maxAllowedContentLength`, For more information, see [IIS Request Limits](#).

New Transport Default Values

The following table describes the settings that have changed and where to find additional information.

PROPERTY	ON	NEW DEFAULT	MORE INFORMATION
channelInitializationTimeout	NetTcpBinding	30 seconds	This property determines how long a TCP connection can take to authenticate itself using the .Net Framing protocol. A client needs to send some initial data before the server has enough information to perform authentication. This timeout is intentionally made smaller than the ReceiveTimeout (10 min) so that malicious unauthenticated clients do not keep the connections tied up to the server for long. The default value is 30 seconds. For more information about ChannelInitializationTimeout
listenBacklog	NetTcpBinding	16 * number of processors	This socket-level property describes the number of "pending accept" requests to be queued. If the listen backlog queue fills up, new socket requests will be rejected. For more information about ListenBacklog
maxPendingAccepts	ConnectionOrientedTransportBindingElement SMSSvcHost.exe	2 * number of processors for transport 4 * number of processors for SMSSvcHost.exe	This property limits the number of channels that the server can have waiting on a listener. When MaxPendingAccepts is too low, there will be a small interval of time in which all of the waiting channels have started servicing connections, but no new channels have begun listening. A connection can arrive during this interval and will fail because nothing is waiting for it on the server. This property can be configured by setting the MaxPendingConnections property to a larger number. For more information, see MaxPendingAccepts and Configuring the Net.TCP Port Sharing Service

PROPERTY	ON	NEW DEFAULT	MORE INFORMATION
maxPendingConnections	ConnectionOrientedTransportBindingElement	12 * number of processors	This property controls how many connections a transport has accepted but have not been picked up by the ServiceModel Dispatcher. To set this value, use <code>MaxConnections</code> on the binding or <code>maxOutboundConnectionsPerEndpoint</code> on the binding element. For more information about MaxPendingConnections
receiveTimeout	SMSSvcHost.exe	30 seconds	This property specifies the timeout for reading the TCP framing data and performing connection dispatching from the underlying connections. This exists to put a cap on the amount of time SMSSvcHost.exe service is kept engaged to read the preamble data from an incoming connection. For more information, see Configuring the Net.TCP Port Sharing Service .

NOTE

These new defaults are used only if you deploy the WCF service on a machine with .NET Framework 4.5. If you deploy the same service on a machine with .NET Framework 4.0, then the .NET Framework 4.0 defaults are used. In such cases it is recommended to configure these settings explicitly.

XmlDictionaryReaderQuotas

[XmlDictionaryReaderQuotas](#) contains configurable quota values for XML dictionary readers which limit the amount of memory utilized by an encoder while creating a message. While these quotas are configurable, the default values have changed to lessen the possibility that a developer will need to set them explicitly. `MaxReceivedMessageSize` quota has not been changed so that it can still limit memory consumption preventing the need for you to deal with the complexity of the [XmlDictionaryReaderQuotas](#). The following table shows the quotas, their new default values and a brief explanation of what each quota is used for.

QUOTA NAME	DEFAULT VALUE	DESCRIPTION
MaxArrayLength	Int32.MaxValue	Gets and sets the maximum allowed array length. This quota limits the maximum size of an array of primitives that the XML reader returns, including byte arrays. This quota does not limit memory consumption in the XML reader itself, but in whatever component that is using the reader. For example, when the DataContractSerializer uses a reader secured with MaxArrayLength , it does not deserialize byte arrays larger than this quota.

QUOTA NAME	DEFAULT VALUE	DESCRIPTION
MaxBytesPerRead	Int32.MaxValue	Gets and sets the maximum allowed bytes returned for each read. This quota limits the number of bytes that are read in a single Read operation when reading the element start tag and its attributes. (In non-streamed cases, the element name itself is not counted against the quota). Having too many XML attributes may use up disproportionate processing time because attribute names have to be checked for uniqueness. MaxBytesPerRead mitigates this threat.
MaxDepth	128 nodes deep	This quota limits the maximum nesting depth of XML elements. MaxDepth interacts with MaxBytesPerRead : the reader always keeps data in memory for the current element and all of its ancestors, so the maximum memory consumption of the reader is proportional to the product of these two settings. When deserializing a deeply-nested object graph, the deserializer is forced to access the entire stack and throw an unrecoverable StackOverflowException . A direct correlation exists between XML nesting and object nesting for both the DataContractSerializer and the XmlSerializer . MaxDepth is used to mitigate this threat.
MaxNameTableCharCount	Int32.MaxValue	This quota limits the maximum number of characters allowed in a nametable. The nametable contains certain strings (such as namespaces and prefixes) that are encountered when processing an XML document. As these strings are buffered in memory, this quota is used to prevent excessive buffering when streaming is expected.
MaxStringContentLength	Int32.MaxValue	This quota limits the maximum string size that the XML reader returns. This quota does not limit memory consumption in the XML reader itself, but in the component that is using the reader. For example, when the DataContractSerializer uses a reader secured with MaxStringContentLength , it does not deserialize strings larger than this quota.

IMPORTANT

Refer to "Using XML Safely" under [Security Considerations for Data](#) for more information about securing your data.

NOTE

These new defaults are used only if you deploy the WCF service on a machine with .NET Framework 4.5. If you deploy the same service on a machine with .NET Framework 4.0, then the .NET Framework 4.0 defaults are used. In such cases it is recommended to configure these settings explicitly.

WCF Configuration Validation

As part of the build process within Visual Studio, WCF configuration files are now validated. A list of validation errors or warnings are displayed in Visual Studio if the validation fails.

XML Editor Tooltips

In order to help new and existing WCF service developers to configure their services, the Visual Studio XML editor now provides tooltips for every configuration element and its properties that is part of the service configuration file.

BasicHttpBinding Improvements

1. Enables a single WCF endpoint to respond to different authentication modes.
2. Enables a WCF service's security settings to be controlled by IIS

Guide to the Documentation

8/31/2018 • 2 minutes to read • [Edit Online](#)

Provided here is guidance about the Windows Communication Foundation (WCF) documentation. The linked documents are recommended starting points grouped according to specific interests and levels of expertise.

To install a stand-alone version of the documentation and a Help viewer, download the [Microsoft Windows SDK v 7.1](#).

New to Windows Communication Foundation Programming

- If you are new to programming with WCF and you just want to see sample applications that work, see the topics listed in [Windows Communication Foundation Samples](#).
- For a tutorial that walks through the basic steps of creating a WCF service and client, see [Getting Started Tutorial](#).
- If you are interested in the concepts behind WCF, see the topics in the [Conceptual Overview](#) section.
- To see graphic examples of client/server security configurations, see [Common Security Scenarios](#).

Programming In-Depth

- If you are ready to start developing an application, see [Basic WCF Programming](#).
- If you are looking for guidance about a particular feature or capability of WCF, see the topics under [WCF Feature Details](#).
- If you would like to extend or customize WCF to suit your requirements, see [Extending WCF](#).
- For information about the tools that help to create and debug WCF applications, see [Windows Communication Foundation Tools](#).
- Configuration using XML files is a primary way of programming WCF services and clients. For reference documentation for the XML elements used in configuration files, see [WCF Configuration Schema](#).

Troubleshooting

For information about troubleshooting common WCF problems, see [WCF Troubleshooting Quickstart](#).

Using Windows Communication Foundation with Other Technologies

- To create a service that communicates with ASP.NET clients, see [How to: Configure WCF Service to Interoperate with ASP.NET Web Service Clients](#).
- Integration with .NET Framework remoting is explained in [Migrating .NET Remoting Applications to WCF](#).
- To integrate an existing COM+ application with a WCF service or client, see [Integrating with COM+ Applications Overview](#).
- To integrate an existing COM application with a WCF service or client, see [Integrating with COM Applications](#).
- To integrate an existing MSMQ application with a WCF service or client, see [How to: Exchange Queued](#)

[Messages with WCF Endpoints](#) and [How to: Exchange Messages with WCF Endpoints and Message Queuing Applications](#).

- To use Internet Information Services (IIS) to host a service, see [Hosting Services](#).
- To use WCF to consume a Web Services Extensions (WSE) 3.0 service, see [How to: Access a WSE 3.0 Service](#).

WS-* Protocols Supported in Windows Communication Foundation

To see a list of protocols supported in the system-provided bindings, see [Web Services Protocols Supported by System-Provided Interoperability Bindings](#). To see the list of system-provided bindings, see [System-Provided Bindings](#).

See Also

[Windows Communication Foundation Samples](#)

[Conceptual Overview](#)

[Guidelines and Best Practices](#)

[Building Clients](#)

Conceptual Overview

5/5/2018 • 2 minutes to read • [Edit Online](#)

This topic summarizes information about the Windows Communication Foundation (WCF) messaging system and the classes that support its use.

In This Section

[What Is Windows Communication Foundation](#)

A brief overview of WCF.

[Fundamental Windows Communication Foundation Concepts](#)

An outline of the major concepts of the WCF programming model.

[Windows Communication Foundation Architecture](#)

A graphical representation of the WCF architecture.

Reference

[System.ServiceModel](#)

Related Sections

[Basic WCF Programming](#)

[Guidelines and Best Practices](#)

[Windows Communication Foundation Samples](#)

[Tools](#)

[System Requirements](#)

[General Reference](#)

What Is Windows Communication Foundation

10/19/2018 • 6 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) is a framework for building service-oriented applications. Using WCF, you can send data as asynchronous messages from one service endpoint to another. A service endpoint can be part of a continuously available service hosted by IIS, or it can be a service hosted in an application. An endpoint can be a client of a service that requests data from a service endpoint. The messages can be as simple as a single character or word sent as XML, or as complex as a stream of binary data. A few sample scenarios include:

- A secure service to process business transactions.
- A service that supplies current data to others, such as a traffic report or other monitoring service.
- A chat service that allows two people to communicate or exchange data in real time.
- A dashboard application that polls one or more services for data and presents it in a logical presentation.
- Exposing a workflow implemented using Windows Workflow Foundation as a WCF service.
- A Silverlight application to poll a service for the latest data feeds.

While creating such applications was possible prior to the existence of WCF, WCF makes the development of endpoints easier than ever. In summary, WCF is designed to offer a manageable approach to creating Web services and Web service clients.

Features of WCF

WCF includes the following set of features. For more information, see [WCF Feature Details](#).

- **Service Orientation**

One consequence of using WS standards is that WCF enables you to create *service oriented* applications. Service-oriented architecture (SOA) is the reliance on Web services to send and receive data. The services have the general advantage of being loosely-coupled instead of hard-coded from one application to another. A loosely-coupled relationship implies that any client created on any platform can connect to any service as long as the essential contracts are met.

- **Interoperability**

WCF implements modern industry standards for Web service interoperability. For more information about the supported standards, see [Interoperability and Integration](#).

- **Multiple Message Patterns**

Messages are exchanged in one of several patterns. The most common pattern is the request/reply pattern, where one endpoint requests data from a second endpoint. The second endpoint replies. There are other patterns such as a one-way message in which a single endpoint sends a message without any expectation of a reply. A more complex pattern is the duplex exchange pattern where two endpoints establish a connection and send data back and forth, similar to an instant messaging program. For more information about how to implement different message exchange patterns using WCF see [Contracts](#).

- **Service Metadata**

WCF supports publishing service metadata using formats specified in industry standards such as WSDL, XML Schema and WS-Policy. This metadata can be used to automatically generate and configure clients for

accessing WCF services. Metadata can be published over HTTP and HTTPS or using the Web Service Metadata Exchange standard. For more information, see [Metadata](#).

- **Data Contracts**

Because WCF is built using the .NET Framework, it also includes code-friendly methods of supplying the contracts you want to enforce. One of the universal types of contracts is the data contract. In essence, as you code your service using Visual C# or Visual Basic, the easiest way to handle data is by creating classes that represent a data entity with properties that belong to the data entity. WCF includes a comprehensive system for working with data in this easy manner. Once you have created the classes that represent data, your service automatically generates the metadata that allows clients to comply with the data types you have designed. For more information, see [Using Data Contracts](#)

- **Security**

Messages can be encrypted to protect privacy and you can require users to authenticate themselves before being allowed to receive messages. Security can be implemented using well-known standards such as SSL or WS-SecureConversation. For more information, see [Security](#).

- **Multiple Transports and Encodings**

Messages can be sent on any of several built-in transport protocols and encodings. The most common protocol and encoding is to send text encoded SOAP messages using the HyperText Transfer Protocol (HTTP) for use on the World Wide Web. Alternatively, WCF allows you to send messages over TCP, named pipes, or MSMQ. These messages can be encoded as text or using an optimized binary format. Binary data can be sent efficiently using the MTOM standard. If none of the provided transports or encodings suit your needs you can create your own custom transport or encoding. For more information about transports and encodings supported by WCF see [Transports](#).

- **Reliable and Queued Messages**

WCF supports reliable message exchange using reliable sessions implemented over WS-Reliable Messaging and using MSMQ. For more information about reliable and queued messaging support in WCF see [Queues and Reliable Sessions](#).

- **Durable Messages**

A durable message is one that is never lost due to a disruption in the communication. The messages in a durable message pattern are always saved to a database. If a disruption occurs, the database allows you to resume the message exchange when the connection is restored. You can also create a durable message using the Windows Workflow Foundation (WF). For more information, see [Workflow Services](#).

- **Transactions**

WCF also supports transactions using one of three transaction models: WS-AtomicTransactions, the APIs in the [System.Transactions](#) namespace, and Microsoft Distributed Transaction Coordinator. For more information about transaction support in WCF see [Transactions](#).

- **AJAX and REST Support**

REST is an example of an evolving Web 2.0 technology. WCF can be configured to process "plain" XML data that is not wrapped in a SOAP envelope. WCF can also be extended to support specific XML formats, such as ATOM (a popular RSS standard), and even non-XML formats, such as JavaScript Object Notation (JSON).

- **Extensibility**

The WCF architecture has a number of extensibility points. If extra capability is required, there are a number of entry points that allow you to customize the behavior of a service. For more information about

available extensibility points see [Extending WCF](#).

WCF Integration with Other Microsoft Technologies

WCF is a flexible platform. Because of this extreme flexibility, WCF is also used in several other Microsoft products. By understanding the basics of WCF, you have an immediate advantage if you also use any of these products.

The first technology to pair with WCF was the Windows Workflow Foundation (WF). Workflows simplify application development by encapsulating steps in the workflow as "activities." In the first version of Windows Workflow Foundation, a developer had to create a host for the workflow. The next version of Windows Workflow Foundation was integrated with WCF. That allowed any workflow to be easily hosted in a WCF service. You can do this by automatically choosing the WF/WCF project type in Visual Studio 2012 or later.

Microsoft BizTalk Server R2 also utilizes WCF as a communication technology. BizTalk is designed to receive and transform data from one standardized format to another. Messages must be delivered to its central message box where the message can be transformed using either a strict mapping or by using one of the BizTalk features such as its workflow engine. BizTalk can now use the WCF Line of Business (LOB) adapter to deliver messages to the message box.

Microsoft Silverlight is a platform for creating interoperable, rich Web applications that allow developers to create media-intensive Web sites (such as streaming video). Beginning with version 2, Silverlight has incorporated WCF as a communication technology to connect Silverlight applications to WCF endpoints.

The hosting features of Windows Server AppFabric application server is specifically built for deploying and managing applications that use WCF for communication. The hosting features includes rich tooling and configuration options specifically designed for WCF-enabled applications.

See Also

- [System.ServiceModel](#)
- [Fundamental Windows Communication Foundation Concepts](#)
- [Windows Communication Foundation Architecture](#)
- [Guidelines and Best Practices](#)
- [Getting Started Tutorial](#)
- [Guide to the Documentation](#)
- [Basic WCF Programming](#)
- [Windows Communication Foundation Samples](#)

Fundamental Windows Communication Foundation Concepts

8/31/2018 • 12 minutes to read • [Edit Online](#)

This document provides a high-level view of the Windows Communication Foundation (WCF) architecture. It is intended to explain key concepts and how they fit together. For a tutorial on creating the simplest version of a WCF service and client, see [Getting Started Tutorial](#). To learn WCF programming, see [Basic WCF Programming](#).

WCF Fundamentals

WCF is a runtime and a set of APIs for creating systems that send messages between services and clients. The same infrastructure and APIs are used to create applications that communicate with other applications on the same computer system or on a system that resides in another company and is accessed over the Internet.

Messaging and Endpoints

WCF is based on the notion of message-based communication, and anything that can be modeled as a message (for example, an HTTP request or a Message Queuing (also known as MSMQ) message) can be represented in a uniform way in the programming model. This enables a unified API across different transport mechanisms.

The model distinguishes between *clients*, which are applications that initiate communication, and *services*, which are applications that wait for clients to communicate with them and respond to that communication. A single application can act as both a client and a service. For examples, see [Duplex Services](#) and [Peer-to-Peer Networking](#).

Messages are sent between endpoints. *Endpoints* are places where messages are sent or received (or both), and they define all the information required for the message exchange. A service exposes one or more application endpoints (as well as zero or more infrastructure endpoints), and the client generates an endpoint that is compatible with one of the service's endpoints.

An *endpoint* describes in a standard-based way where messages should be sent, how they should be sent, and what the messages should look like. A service can expose this information as metadata that clients can process to generate appropriate WCF clients and communication *stacks*.

Communication Protocols

One required element of the communication stack is the *transport protocol*. Messages can be sent over intranets and the Internet using common transports, such as HTTP and TCP. Other transports are included that support communication with Message Queuing applications and nodes on a Peer Networking mesh. More transport mechanisms can be added using the built-in extension points of WCF.

Another required element in the communication stack is the encoding that specifies how any given message is formatted. WCF provides the following encodings:

- Text encoding, an interoperable encoding.
- Message Transmission Optimization Mechanism (MTOM) encoding, which is an interoperable way for efficiently sending unstructured binary data to and from a service.
- Binary encoding for efficient transfer.

More encoding mechanisms (for example, a compression encoding) can be added using the built-in extension points of WCF.

Message Patterns

WCF supports several messaging patterns, including request-reply, one-way, and duplex communication. Different transports support different messaging patterns, and thus affect the types of interactions that they support. The WCF APIs and runtime also help you to send messages securely and reliably.

WCF Terms

Other concepts and terms used in the WCF documentation include the following.

message

A self-contained unit of data that can consist of several parts, including a body and headers.

service

A construct that exposes one or more endpoints, with each endpoint exposing one or more service operations.

endpoint

A construct at which messages are sent or received (or both). It comprises a location (an address) that defines where messages can be sent, a specification of the communication mechanism (a binding) that describes how messages should be sent, and a definition for a set of messages that can be sent or received (or both) at that location (a service contract) that describes what message can be sent.

A WCF service is exposed to the world as a collection of endpoints.

application endpoint

An endpoint exposed by the application and that corresponds to a service contract implemented by the application.

infrastructure endpoint

An endpoint that is exposed by the infrastructure to facilitate functionality that is needed or provided by the service that does not relate to a service contract. For example, a service might have an infrastructure endpoint that provides metadata information.

address

Specifies the location where messages are received. It is specified as a Uniform Resource Identifier (URI). The URI schema part names the transport mechanism to use to reach the address, such as HTTP and TCP. The hierarchical part of the URI contains a unique location whose format is dependent on the transport mechanism.

The endpoint address enables you to create unique endpoint addresses for each endpoint in a service or, under certain conditions, to share an address across endpoints. The following example shows an address using the HTTPS protocol with a non-default port:

```
HTTPS://cohowinery:8005/ServiceModelSamples/CalculatorService
```

binding

Defines how an endpoint communicates to the world. It is constructed of a set of components called binding elements that "stack" one on top of the other to create the communication infrastructure. At the very least, a binding defines the transport (such as HTTP or TCP) and the encoding being used (such as text or binary). A binding can contain binding elements that specify details like the security mechanisms used to secure messages, or the message pattern used by an endpoint. For more information, see [Configuring Services](#).

binding element

Represents a particular piece of the binding, such as a transport, an encoding, an implementation of an infrastructure-level protocol (such as WS-ReliableMessaging), or any other component of the communication stack.

behaviors

A component that controls various run-time aspects of a service, an endpoint, a particular operation, or a client.

Behaviors are grouped according to scope: common behaviors affect all endpoints globally, service behaviors affect only service-related aspects, endpoint behaviors affect only endpoint-related properties, and operation-level behaviors affect particular operations. For example, one service behavior is throttling, which specifies how a service reacts when an excess of messages threaten to overwhelm its handling capabilities. An endpoint behavior, on the other hand, controls only aspects that are relevant to endpoints, such as how and where to find a security credential.

system-provided bindings

WCF includes a number of system-provided bindings. These are collections of binding elements that are optimized for specific scenarios. For example, the [WSHttpBinding](#) is designed for interoperability with services that implement various WS-* specifications. These predefined bindings save time by presenting only those options that can be correctly applied to the specific scenario. If a predefined binding does not meet your requirements, you can create your own custom binding.

configuration versus coding

Control of an application can be done either through coding, through configuration, or through a combination of both. Configuration has the advantage of allowing someone other than the developer (for example, a network administrator) to set client and service parameters after the code is written and without having to recompile. Configuration not only enables you to set values like endpoint addresses, but also allows further control by enabling you to add endpoints, bindings, and behaviors. Coding allows the developer to retain strict control over all components of the service or client, and any settings done through the configuration can be inspected and if needed overridden by the code.

service operation

A procedure defined in a service's code that implements the functionality for an operation. This operation is exposed to clients as methods on a WCF client. The method can return a value, and can take an optional number of arguments, or take no arguments, and return no response. For example, an operation that functions as a simple "Hello" can be used as a notification of a client's presence and to begin a series of operations.

service contract

Ties together multiple related operations into a single functional unit. The contract can define service-level settings, such as the namespace of the service, a corresponding callback contract, and other such settings. In most cases, the contract is defined by creating an interface in the programming language of your choice and applying the [ServiceContractAttribute](#) attribute to the interface. The actual service code results by implementing the interface.

operation contract

An operation contract defines the parameters and return type of an operation. When creating an interface that defines the service contract, you signify an operation contract by applying the [OperationContractAttribute](#) attribute to each method definition that is part of the contract. The operations can be modeled as taking a single message and returning a single message, or as taking a set of types and returning a type. In the latter case, the system will determine the format for the messages that need to be exchanged for that operation.

message contract

Describes the format of a message. For example, it declares whether message elements should go in headers versus the body, what level of security should be applied to what elements of the message, and so on.

fault contract

Can be associated with a service operation to denote errors that can be returned to the caller. An operation can have zero or more faults associated with it. These errors are SOAP faults that are modeled as exceptions in the programming model.

data contract

The descriptions in metadata of the data types that a service uses. This enables others to interoperate with the service. The data types can be used in any part of a message, for example, as parameters or return types. If the service is using only simple types, there is no need to explicitly use data contracts.

hosting

A service must be hosted in some process. A *host* is an application that controls the lifetime of the service. Services can be self-hosted or managed by an existing hosting process.

self-hosted service

A service that runs within a process application that the developer created. The developer controls its lifetime, sets the properties of the service, opens the service (which sets it into a listening mode), and closes the service.

hosting process

An application that is designed to host services. These include Internet Information Services (IIS), Windows Activation Services (WAS), and Windows Services. In these hosted scenarios, the host controls the lifetime of the service. For example, using IIS you can set up a virtual directory that contains the service assembly and configuration file. When a message is received, IIS starts the service and controls its lifetime.

instanting

A service has an instancing model. There are three instancing models: "single," in which a single CLR object services all the clients; "per call," in which a new CLR object is created to handle each client call; and "per session," in which a set of CLR objects is created, one for each separate session. The choice of an instancing model depends on the application requirements and the expected usage pattern of the service.

client application

A program that exchanges messages with one or more endpoints. The client application begins by creating an instance of a WCF client and calling methods of the WCF client. It is important to note that a single application can be both a client and a service.

channel

A concrete implementation of a binding element. The binding represents the configuration, and the channel is the implementation associated with that configuration. Therefore, there is a channel associated with each binding element. Channels stack on top of each other to create the concrete implementation of the binding: the channel stack.

WCF client

A client-application construct that exposes the service operations as methods (in the .NET Framework programming language of your choice, such as Visual Basic or Visual C#). Any application can host a WCF client, including an application that hosts a service. Therefore, it is possible to create a service that includes WCF clients of other services.

A WCF client can be automatically generated by using the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) and pointing it at a running service that publishes metadata.

metadata

In a service, describes the characteristics of the service that an external entity needs to understand to communicate with the service. Metadata can be consumed by the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) to generate a WCF client and accompanying configuration that a client application can use to interact with the service.

The metadata exposed by the service includes XML schema documents, which define the data contract of the service, and WSDL documents, which describe the methods of the service.

When enabled, metadata for the service is automatically generated by WCF by inspecting the service and its endpoints. To publish metadata from a service, you must explicitly enable the metadata behavior.

security

In WCF, includes confidentiality (encryption of messages to prevent eavesdropping), integrity (the means for detection of tampering with the message), authentication (the means for validation of servers and clients), and authorization (the control of access to resources). These functions are provided by either leveraging existing

security mechanisms, such as TLS over HTTP (also known as HTTPS), or by implementing one or more of the various WS-* security specifications.

transport security mode

Specifies that confidentiality, integrity, and authentication are provided by the transport layer mechanisms (such as HTTPS). When using a transport like HTTPS, this mode has the advantage of being efficient in its performance, and well understood because of its prevalence on the Internet. The disadvantage is that this kind of security is applied separately on each hop in the communication path, making the communication susceptible to a "man in the middle" attack.

message security mode

Specifies that security is provided by implementing one or more of the security specifications, such as the specification named [Web Services Security: SOAP Message Security](#). Each message contains the necessary mechanisms to provide security during its transit, and to enable the receivers to detect tampering and to decrypt the messages. In this sense, the security is encapsulated within every message, providing end-to-end security across multiple hops. Because security information becomes part of the message, it is also possible to include multiple kinds of credentials with the message (these are referred to as *claims*). This approach also has the advantage of enabling the message to travel securely over any transport, including multiple transports between its origin and destination. The disadvantage of this approach is the complexity of the cryptographic mechanisms employed, resulting in performance implications.

transport with message credential security mode

Specifies the use of the transport layer to provide confidentiality, authentication, and integrity of the messages, while each of the messages can contain multiple credentials (claims) required by the receivers of the message.

WS-*

Shorthand for the growing set of Web Service (WS) specifications, such as WS-Security, WS-ReliableMessaging, and so on, that are implemented in WCF.

See Also

[What Is Windows Communication Foundation](#)

[Windows Communication Foundation Architecture](#)

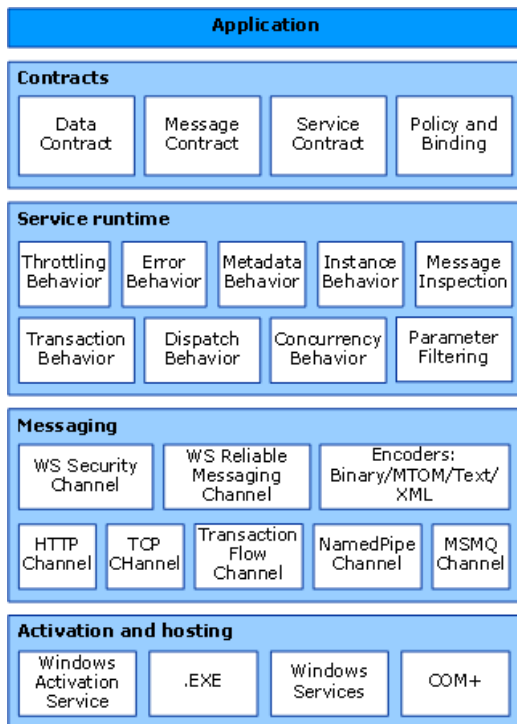
[Security Architecture](#)

Windows Communication Foundation Architecture

5/5/2018 • 3 minutes to read • [Edit Online](#)

The following graphic illustrates the major layers of the Windows Communication Foundation (WCF) architecture.

WCF Architecture



Contracts and Descriptions

Contracts define various aspects of the message system. The data contract describes every parameter that makes up every message that a service can create or consume. The message parameters are defined by XML Schema definition language (XSD) documents, enabling any system that understands XML to process the documents. The message contract defines specific message parts using SOAP protocols, and allows finer-grained control over parts of the message, when interoperability demands such precision. The service contract specifies the actual method signatures of the service, and is distributed as an interface in one of the supported programming languages, such as Visual Basic or Visual C#.

Policies and bindings stipulate the conditions required to communicate with a service. For example, the binding must (at a minimum) specify the transport used (for example, HTTP or TCP), and an encoding. Policies include security requirements and other conditions that must be met to communicate with a service.

Service Runtime

The service runtime layer contains the behaviors that occur only during the actual operation of the service, that is, the runtime behaviors of the service. Throttling controls how many messages are processed, which can be varied if the demand for the service grows to a preset limit. An error behavior specifies what occurs when an internal error occurs on the service, for example, by controlling what information is communicated to the client. (Too much information can give a malicious user an advantage in mounting an attack.) Metadata behavior governs how and whether metadata is made available to the outside world. Instance behavior specifies how many instances of the service can be run (for example, a singleton specifies only one instance to process all messages). Transaction behavior enables the rollback of transacted operations if a failure occurs. Dispatch behavior is the control of how a message is processed by the WCF infrastructure.

Extensibility enables customization of runtime processes. For example, message inspection is the facility to inspect parts of a message, and parameter filtering enables preset actions to occur based on filters acting on message headers.

Messaging

The messaging layer is composed of *channels*. A channel is a component that processes a message in some way, for example, by authenticating a message. A set of channels is also known as a *channel stack*. Channels operate on messages and message headers. This is different from the service runtime layer, which is primarily concerned about processing the contents of message bodies.

There are two types of channels: transport channels and protocol channels.

Transport channels read and write messages from the network (or some other communication point with the outside world). Some transports use an encoder to convert messages (which are represented as XML Infosets) to and from the byte stream representation used by the network. Examples of transports are HTTP, named pipes, TCP, and MSMQ. Examples of encodings are XML and optimized binary.

Protocol channels implement message processing protocols, often by reading or writing additional headers to the message. Examples of such protocols include WS-Security and WS-Reliability.

The messaging layer illustrates the possible formats and exchange patterns of the data. WS-Security is an implementation of the WS-Security specification enabling security at the message layer. The WS-Reliable Messaging channel enables the guarantee of message delivery. The encoders present a variety of encodings that can be used to suit the needs of the message. The HTTP channel specifies that the HyperText Transport Protocol is used for message delivery. The TCP channel similarly specifies the TCP protocol. The Transaction Flow channel governs transacted message patterns. The Named Pipe channel enables interprocess communication. The MSMQ channel enables interoperation with MSMQ applications.

Hosting and Activation

In its final form, a service is a program. Like other programs, a service must be run in an executable. This is known as a *self-hosted* service.

Services can also be *hosted*, or run in an executable managed by an external agent, such as IIS or Windows Activation Service (WAS). WAS enables WCF applications to be activated automatically when deployed on a computer running WAS. Services can also be manually run as executables (.exe files). A service can also be run automatically as a Windows service. COM+ components can also be hosted as WCF services.

See Also

[What Is Windows Communication Foundation](#)

[Fundamental Windows Communication Foundation Concepts](#)

WCF and .NET Framework Client Profile

5/4/2018 • 2 minutes to read • [Edit Online](#)

.NET Framework Client Profile is a lightweight version of the full .Net Framework designed for clients that don't need the entire framework. Not all of Windows Communication Foundation is supported by the client framework.

WCF features supported by the .Net Framework Client Profile

The following Windows Communication Foundation features are supported by .NET Framework Client Profile:

- All of WCF is supported except for Cardspace and web hosting.
- Remoting TCP/IP channels are supported.
- Asmx (Web Services) are not supported.

Getting Started Tutorial

10/3/2018 • 3 minutes to read • [Edit Online](#)

The topics contained in this section are intended to give you quick exposure to the Windows Communication Foundation (WCF) programming experience. They are designed to be completed in the order of the list at the bottom of this topic. Working through this tutorial gives you an introductory understanding of the steps required to create WCF service and client applications. A service exposes one or more endpoints, each of which exposes one or more service operations. The *endpoint* of a service specifies an address where the service can be found, a binding that contains the information that describes how a client must communicate with the service, and a contract that defines the functionality provided by the service to its clients.

After you work through the sequence of topics in this tutorial, you will have a running service, and a client that calls the service. The first three topics describe how to define a service contract, how to implement the service contract, and how to host the service. The service that is created is self-hosted within a console application. Services can also be hosted under Internet Information Services (IIS). For more information about how to do this, see [How to: Host a WCF Service in IIS](#). The service is configured in code; however, services can also be configured within a configuration file. For more information about using a configuration file see [Configuring Services Using Configuration Files](#).

The next three topics describe how to create a client proxy, configure the client application, and use the client proxy to call service operation exposed by the service. Services publish metadata that define the information a client application needs to communicate with the service. Visual Studio 2012 automates the process of accessing this metadata and uses it to construct and configure the client application for the service. If you are not using Visual Studio 2012, you can use the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) to construct and configure the client application for the service.

The topics in this section assume you are using Visual Studio as the development environment. If you are using another development environment, ignore the Visual Studio-specific instructions.

For sample applications that can be downloaded to your hard disk and run, see the topics in [Windows Communication Foundation Samples](#). For this topic, see, in particular, the [Getting Started](#).

For more in-depth information about creating services and clients, see [Basic WCF Programming](#).

In This Section

[How to: Define a Service Contract](#)

Describes how to create a WCF contract using a user-defined interface. The contract defines the functionality exposed by the service.

[How to: Implement a Service Contract](#)

Describes how to implement a service contract. Once a contract is define, it must be implemented with a service class.

[How to: Host and Run a Basic Service](#)

Describes how to configure an endpoint for the service in code and how to host the service in a console application. To become active, a service must be configured and hosted within a run-time environment. This environment creates the service and controls its context and lifetime.

[How to: Create a Client](#)

Describes how to retrieve metadata used to create a WCF client proxy from a WCF service. This process uses the Add Service Reference functionality within Visual Studio.

[How to: Configure a Client](#)

Describes how to configure a WCF client. Configuring the client requires specifying the endpoint that the client uses to access the service.

[How to: Use a Client](#)

Describes how to use the WCF client proxy to invoke service operations.

Reference

- [ServiceContractAttribute](#)
- [OperationContractAttribute](#)

Related Sections

- [Windows Communication Foundation Samples](#)
- [Basic Programming Lifecycle](#)

See Also

- [Conceptual Overview](#)
- [Guide to the Documentation](#)
- [What Is Windows Communication Foundation](#)
- [WCF Feature Details](#)

How to: Define a Windows Communication Foundation Service Contract

10/3/2018 • 2 minutes to read • [Edit Online](#)

This is the first of six tasks required to create a basic Windows Communication Foundation (WCF) application. For an overview of all six of the tasks, see the [Getting Started Tutorial](#) topic.

When creating a WCF service, the first task is to define a service contract. The service contract specifies what operations the service supports. An operation can be thought of as a Web service method. Contracts are created by defining a C++, C#, or Visual Basic (VB) interface. Each method in the interface corresponds to a specific service operation. Each interface must have the [ServiceContractAttribute](#) applied to it and each operation must have the [OperationContractAttribute](#) attribute applied to it. If a method within an interface that has the [ServiceContractAttribute](#) attribute does not have the [OperationContractAttribute](#) attribute, that method is not exposed by the service.

The code used for this task is provided in the example following the procedure.

Define a service contract

1. Open Visual Studio as an administrator by right-clicking the program in the **Start** menu and selecting **More > Run as administrator**.
2. Create a WCF Service Library project.
 - a. From the **File** menu, select **New > Project**.
 - b. In the **New Project** dialog, on the left-hand side, expand **Visual C#** or **Visual Basic**, and then select the **WCF** category. A list of project templates is displayed in the center section of the dialog. Select **WCF Service Library**.
 - c. Enter `GettingStartedLib` in the **Name** textbox and `GettingStarted` in the **Solution name** textbox at the bottom of the dialog.

NOTE

If you don't see the **WCF** project template category, you may need to install the **Windows Communication Foundation** component of Visual Studio. In the **New Project** dialog box, click the link that says **Open Visual Studio Installer**. Select the **Individual Components** tab, and then find and select **Windows Communication Foundation** under the **Development activities** category. Choose **Modify** to begin installing the component.

Visual Studio creates the project, which contains 3 files: `IService1.cs` (or `IService1.vb`), `Service1.cs` (or `Service1.vb`), and `App.config`. The `IService1` file contains a default service contract. The `Service1` file contains a default implementation of the service contract. The `App.config` file contains configuration needed to load the default service with the Visual Studio WCF Service Host. For more information about the WCF Service Host tool, see [WCF Service Host \(WcfSvcHost.exe\)](#)

3. Open the `IService1.cs` or `IService1.vb` file and delete the code within the namespace declaration, leaving the namespace declaration. Inside the namespace declaration define a new interface called `ICalculator` as shown in the code below.

```

using System;
using System.ServiceModel;

namespace GettingStartedLib
{
    [ServiceContract(Namespace = "http://Microsoft.ServiceModel.Samples")]
    public interface ICalculator
    {
        [OperationContract]
        double Add(double n1, double n2);
        [OperationContract]
        double Subtract(double n1, double n2);
        [OperationContract]
        double Multiply(double n1, double n2);
        [OperationContract]
        double Divide(double n1, double n2);
    }
}

```

```

Imports System.ServiceModel

Namespace GettingStartedLib

    <ServiceContract(Namespace:="http://Microsoft.ServiceModel.Samples")> _
    Public Interface ICalculator

        <OperationContract()> _
        Function Add(ByVal n1 As Double, ByVal n2 As Double) As Double
        <OperationContract()> _
        Function Subtract(ByVal n1 As Double, ByVal n2 As Double) As Double
        <OperationContract()> _
        Function Multiply(ByVal n1 As Double, ByVal n2 As Double) As Double
        <OperationContract()> _
        Function Divide(ByVal n1 As Double, ByVal n2 As Double) As Double
    End Interface
End Namespace

```

This contract defines an online calculator. Notice the `ICalculator` interface is marked with the [ServiceContractAttribute](#) attribute. This attribute defines a namespace that is used to disambiguate the contract name. Each calculator operation is marked with the [OperationContractAttribute](#) attribute.

Next steps

[How to: Implement a service contract](#)

See also

- [ServiceContractAttribute](#)
- [OperationContractAttribute](#)
- [How to: Implement a Service Contract](#)
- [Getting Started](#)
- [Self-Host](#)

How to: Implement a Windows Communication Foundation Service Contract

9/18/2018 • 4 minutes to read • [Edit Online](#)

This is the second of six tasks required to create a basic Windows Communication Foundation (WCF) service and a client that can call the service. For an overview of all six tasks, see the [Getting Started Tutorial](#) topic.

The next step in creating a WCF application is to implement the service interface. This involves creating a class called `CalculatorService` that implements the user-defined `ICalculator` interface..

To implement a WCF service contract

Open the `Service1.cs` or `Service1.vb` file and add the following code:

```
using System;
using System.ServiceModel;

namespace GettingStartedLib
{
    public class CalculatorService : ICalculator
    {
        public double Add(double n1, double n2)
        {
            double result = n1 + n2;
            Console.WriteLine("Received Add({0},{1})", n1, n2);
            // Code added to write output to the console window.
            Console.WriteLine("Return: {0}", result);
            return result;
        }

        public double Subtract(double n1, double n2)
        {
            double result = n1 - n2;
            Console.WriteLine("Received Subtract({0},{1})", n1, n2);
            Console.WriteLine("Return: {0}", result);
            return result;
        }

        public double Multiply(double n1, double n2)
        {
            double result = n1 * n2;
            Console.WriteLine("Received Multiply({0},{1})", n1, n2);
            Console.WriteLine("Return: {0}", result);
            return result;
        }

        public double Divide(double n1, double n2)
        {
            double result = n1 / n2;
            Console.WriteLine("Received Divide({0},{1})", n1, n2);
            Console.WriteLine("Return: {0}", result);
            return result;
        }
    }
}
```

```
Imports System.ServiceModel

Namespace GettingStartedLib

    Public Class CalculatorService
        Implements ICalculator

        Public Function Add(ByVal n1 As Double, ByVal n2 As Double) As Double Implements ICalculator.Add
            Dim result As Double = n1 + n2
            ' Code added to write output to the console window.
            Console.WriteLine("Received Add({0},{1})", n1, n2)
            Console.WriteLine("Return: {0}", result)
            Return result
        End Function

        Public Function Subtract(ByVal n1 As Double, ByVal n2 As Double) As Double Implements
ICalculator.Subtract
            Dim result As Double = n1 - n2
            Console.WriteLine("Received Subtract({0},{1})", n1, n2)
            Console.WriteLine("Return: {0}", result)
            Return result
        End Function

        Public Function Multiply(ByVal n1 As Double, ByVal n2 As Double) As Double Implements
ICalculator.Multiply
            Dim result As Double = n1 * n2
            Console.WriteLine("Received Multiply({0},{1})", n1, n2)
            Console.WriteLine("Return: {0}", result)
            Return result
        End Function

        Public Function Divide(ByVal n1 As Double, ByVal n2 As Double) As Double Implements ICalculator.Divide
            Dim result As Double = n1 / n2
            Console.WriteLine("Received Divide({0},{1})", n1, n2)
            Console.WriteLine("Return: {0}", result)
            Return result
        End Function
    End Class
End Namespace
```

Each method implements the calculator operation and writes some text to the console to make testing easier.

Example

The following code shows both the interface that defines the contract and the implementation of the interface.

```

using System;
using System.ServiceModel;

namespace GettingStartedLib
{
    [ServiceContract(Namespace = "http://Microsoft.ServiceModel.Samples")]
    public interface ICalculator
    {
        [OperationContract]
        double Add(double n1, double n2);
        [OperationContract]
        double Subtract(double n1, double n2);
        [OperationContract]
        double Multiply(double n1, double n2);
        [OperationContract]
        double Divide(double n1, double n2);
    }
}

```

```

using System;
using System.ServiceModel;

namespace GettingStartedLib
{
    public class CalculatorService : ICalculator
    {
        public double Add(double n1, double n2)
        {
            double result = n1 + n2;
            Console.WriteLine("Received Add({0},{1})", n1, n2);
            // Code added to write output to the console window.
            Console.WriteLine("Return: {0}", result);
            return result;
        }

        public double Subtract(double n1, double n2)
        {
            double result = n1 - n2;
            Console.WriteLine("Received Subtract({0},{1})", n1, n2);
            Console.WriteLine("Return: {0}", result);
            return result;
        }

        public double Multiply(double n1, double n2)
        {
            double result = n1 * n2;
            Console.WriteLine("Received Multiply({0},{1})", n1, n2);
            Console.WriteLine("Return: {0}", result);
            return result;
        }

        public double Divide(double n1, double n2)
        {
            double result = n1 / n2;
            Console.WriteLine("Received Divide({0},{1})", n1, n2);
            Console.WriteLine("Return: {0}", result);
            return result;
        }
    }
}

```

```
Imports System.ServiceModel

Namespace GettingStartedLib

    <ServiceContract(Namespace:="http://Microsoft.ServiceModel.Samples")> _
    Public Interface ICalculator

        <OperationContract> _
        Function Add(ByVal n1 As Double, ByVal n2 As Double) As Double
        <OperationContract> _
        Function Subtract(ByVal n1 As Double, ByVal n2 As Double) As Double
        <OperationContract> _
        Function Multiply(ByVal n1 As Double, ByVal n2 As Double) As Double
        <OperationContract> _
        Function Divide(ByVal n1 As Double, ByVal n2 As Double) As Double

    End Interface
End Namespace
```

```
Imports System.ServiceModel

Namespace GettingStartedLib

    Public Class CalculatorService
        Implements ICalculator

        Public Function Add(ByVal n1 As Double, ByVal n2 As Double) As Double Implements ICalculator.Add
            Dim result As Double = n1 + n2
            ' Code added to write output to the console window.
            Console.WriteLine("Received Add({0},{1})", n1, n2)
            Console.WriteLine("Return: {0}", result)
            Return result
        End Function

        Public Function Subtract(ByVal n1 As Double, ByVal n2 As Double) As Double Implements
        ICalculator.Subtract
            Dim result As Double = n1 - n2
            Console.WriteLine("Received Subtract({0},{1})", n1, n2)
            Console.WriteLine("Return: {0}", result)
            Return result

        End Function

        Public Function Multiply(ByVal n1 As Double, ByVal n2 As Double) As Double Implements
        ICalculator.Multiply
            Dim result As Double = n1 * n2
            Console.WriteLine("Received Multiply({0},{1})", n1, n2)
            Console.WriteLine("Return: {0}", result)
            Return result

        End Function

        Public Function Divide(ByVal n1 As Double, ByVal n2 As Double) As Double Implements ICalculator.Divide
            Dim result As Double = n1 / n2
            Console.WriteLine("Received Divide({0},{1})", n1, n2)
            Console.WriteLine("Return: {0}", result)
            Return result

        End Function
    End Class
End Namespace
```

Compile the code

Build the solution to ensure there are no compilation errors. If you're using Visual Studio, on the **Build** menu

select **Build Solution** (or press **Ctrl+Shift+B**).

Next steps

Now the service contract is created and implemented. In the next step, you run the service.

[How to: Host and Run a Basic Service](#)

For troubleshooting information, see [Troubleshooting the Getting Started Tutorial](#).

See also

- [Getting Started](#)
- [Self-Host](#)

How to: Host and Run a Basic Windows Communication Foundation Service

11/13/2018 • 9 minutes to read • [Edit Online](#)

This is the third of six tasks required to create a Windows Communication Foundation (WCF) application. For an overview of all six of the tasks, see the [Getting Started Tutorial](#) topic.

This topic describes how to host a Windows Communication Foundation (WCF) service in a console application. This procedure consists of the following steps:

- Create a console application project to host the service.
- Create a service host for the service.
- Enable metadata exchange.
- Open the service host.

A complete listing of the code written in this task is provided in the example following the procedure.

Create a new console application to host the service

1. Create a new Console Application project in Visual Studio by right-clicking on the Getting Started solution and selecting **Add > New Project**. In the **Add New Project** dialog, on the left-hand side, select the **Windows Desktop** category under **Visual C#** or **Visual Basic**. Select the **Console App (.NET Framework)** template, and then name the project **GettingStartedHost**.
2. Add a reference to the GettingStartedLib project to the GettingStartedHost project. Right-click on the **References** folder under the GettingStartedHost project in **Solution Explorer**, and then select **Add Reference**. In the **Add Reference** dialog, select **Solution** on the left-hand side of the dialog, select GettingStartedLib in the center section of the dialog, and then choose **Add**. This makes the types defined in GettingStartedLib available to the GettingStartedHost project.
3. Add a reference to System.ServiceModel to the GettingStartedHost project. Right-click the **References** folder under the GettingStartedHost project in **Solution Explorer** and select **Add Reference**. In the **Add Reference** dialog, select **Framework** on the left-hand side of the dialog under **Assemblies**. Find and select **System.ServiceModel**, and then choose **OK**. Save the solution by selecting **File > Save All**.

Host the service

Open the Program.cs or Module.vb file and enter the following code:

```

using System;
using System.ServiceModel;
using System.ServiceModel.Description;
using GettingStartedLib;

namespace GettingStartedHost
{
    class Program
    {
        static void Main(string[] args)
        {
            // Step 1 Create a URI to serve as the base address.
            Uri baseAddress = new Uri("http://localhost:8000/GettingStarted/");

            // Step 2 Create a ServiceHost instance
            ServiceHost selfHost = new ServiceHost(typeof(CalculatorService), baseAddress);

            try
            {
                // Step 3 Add a service endpoint.
                selfHost.AddServiceEndpoint(typeof(ICalculator), new WSHttpBinding(), "CalculatorService");

                // Step 4 Enable metadata exchange.
                ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
                smb.HttpGetEnabled = true;
                selfHost.Description.Behaviors.Add(smb);

                // Step 5 Start the service.
                selfHost.Open();
                Console.WriteLine("The service is ready.");
                Console.WriteLine("Press <ENTER> to terminate service.");
                Console.WriteLine();
                Console.ReadLine();

                // Close the ServiceHostBase to shutdown the service.
                selfHost.Close();
            }
            catch (CommunicationException ce)
            {
                Console.WriteLine("An exception occurred: {0}", ce.Message);
                selfHost.Abort();
            }
        }
    }
}

```

```
Imports System.ServiceModel
Imports System.ServiceModel.Description
Imports GettingStartedLibVB.GettingStartedLib

Module Service

    Class Program
        Shared Sub Main()
            ' Step 1 Create a Uri to serve as the base address
            Dim baseAddress As New Uri("http://localhost:8000/ServiceModelSamples/Service")

            ' Step 2 Create a ServiceHost instance
            Dim selfHost As New ServiceHost(GetType(CalculatorService), baseAddress)
            Try

                ' Step 3 Add a service endpoint
                ' Add a service endpoint
                selfHost.AddServiceEndpoint( _
                    GetType(ICalculator), _
                    New WSHttpBinding(), _
                    "CalculatorService")

                ' Step 4 Enable metadata exchange.
                Dim smb As New ServiceMetadataBehavior()
                smb.HttpGetEnabled = True
                selfHost.Description.Behaviors.Add(smb)

                ' Step 5 Start the service
                selfHost.Open()
                Console.WriteLine("The service is ready.")
                Console.WriteLine("Press <ENTER> to terminate service.")
                Console.WriteLine()
                Console.ReadLine()

                ' Close the ServiceHostBase to shutdown the service.
                selfHost.Close()
            Catch ce As CommunicationException
                Console.WriteLine("An exception occurred: {0}", ce.Message)
                selfHost.Abort()
            End Try
        End Sub
    End Class

End Module
```

Step 1 - Creates an instance of the `Uri` class to hold the base address of the service. Services are identified by a URL which contains a base address and an optional URI. The base address is formatted as follows: `[transport]://[machine-name or domain][:optional port #]/[optional URI segment]` The base address for the calculator service uses the HTTP transport, localhost, port 8000, and the URI segment "GettingStarted"

Step 2 – Creates an instance of the [ServiceHost](#) class to host the service. The constructor takes two parameters, the type of the class that implements the service contract, and the base address of the service.

Step 3 – Creates a [ServiceEndpoint](#) instance. A service endpoint is composed of an address, a binding, and a service contract. The [ServiceEndpoint](#) constructor therefore takes the service contract interface type, a binding, and an address. The service contract is `ICalculator`, which you defined and implement in the service type. The binding used in this sample is [WSHttpBinding](#) which is a built-in binding that is used for connecting to endpoints that conform to the WS-* specifications. For more information about WCF bindings, see [WCF Bindings Overview](#). The address is appended to the base address to identify the endpoint. The address specified in this code is "CalculatorService" so the fully qualified address for the endpoint is

```
"http://localhost:8000/GettingStarted/CalculatorService".
```

```
> [!IMPORTANT]
> Adding a service endpoint is optional when using .NET Framework 4 or later. In these versions, if no
endpoints are added in code or configuration, WCF adds one default endpoint for each combination of base
address and contract implemented by the service. For more information about default endpoints see [Specifying
an Endpoint Address](../../docs/framework/wcf/specifying-an-endpoint-address.md). For more information
about default endpoints, bindings, and behaviors, see [Simplified Configuration]
(../../docs/framework/wcf/simplified-configuration.md) and [Simplified Configuration for WCF Services]
(../../docs/framework/wcf/samples/simplified-configuration-for-wcf-services.md).
```

Step 4 – Enable metadata exchange. Clients will use metadata exchange to generate proxies that will be used to call the service operations. To enable metadata exchange create a [ServiceMetadataBehavior](#) instance, set its [HttpGetEnabled](#) property to `true`, and add the behavior to the `System.ServiceModel.ServiceHost.Behaviors` collection of the [ServiceHost](#) instance.

Step 5 – Open the [ServiceHost](#) to listen for incoming messages. Notice the code waits for the user to hit enter. If you do not do this, the app will close immediately and the service will shut down. Also notice a try/catch block used. After the [ServiceHost](#) has been instantiated, all other code is placed in a try/catch block. For more information about safely catching exceptions thrown by [ServiceHost](#), see [Use Close and Abort to release WCF client resources](#)

IMPORTANT

Edit App.config in GettingStartedLib to reflect the changes made in code:

1. Change line 14 to `<service name="GettingStartedLib.CalculatorService">`
2. Change line 17 to `<add baseAddress = "http://localhost:8000/GettingStarted/CalculatorService" />`
3. Change line 22 to `<endpoint address="" binding="wsHttpBinding" contract="GettingStartedLib.ICalculator">`

Verify the service is working

1. Run the GettingStartedHost console application from inside Visual Studio.

The service must be run with administrator privileges. Because Visual Studio was opened with administrator privileges, GettingStartedHost is also run with administrator privileges. You can also open a new command prompt using **Run as administrator** and run service.exe within it.

2. Open a web browser and browse to the service's debug page at

`http://localhost:8000/GettingStarted/CalculatorService`.

Example

The following example includes the service contract and implementation from previous steps in the tutorial and hosts the service in a console application.

To compile this with a command-line compiler, compile IService1.cs and Service1.cs into a class library that references `System.ServiceModel.dll`. Compile Program.cs as a console application.

```

using System;
using System.ServiceModel;

namespace GettingStartedLib
{
    [ServiceContract(Namespace = "http://Microsoft.ServiceModel.Samples")]
    public interface ICalculator
    {
        [OperationContract]
        double Add(double n1, double n2);
        [OperationContract]
        double Subtract(double n1, double n2);
        [OperationContract]
        double Multiply(double n1, double n2);
        [OperationContract]
        double Divide(double n1, double n2);
    }
}

```

```

using System;
using System.ServiceModel;

namespace GettingStartedLib
{
    public class CalculatorService : ICalculator
    {
        public double Add(double n1, double n2)
        {
            double result = n1 + n2;
            Console.WriteLine("Received Add({0},{1})", n1, n2);
            // Code added to write output to the console window.
            Console.WriteLine("Return: {0}", result);
            return result;
        }

        public double Subtract(double n1, double n2)
        {
            double result = n1 - n2;
            Console.WriteLine("Received Subtract({0},{1})", n1, n2);
            Console.WriteLine("Return: {0}", result);
            return result;
        }

        public double Multiply(double n1, double n2)
        {
            double result = n1 * n2;
            Console.WriteLine("Received Multiply({0},{1})", n1, n2);
            Console.WriteLine("Return: {0}", result);
            return result;
        }

        public double Divide(double n1, double n2)
        {
            double result = n1 / n2;
            Console.WriteLine("Received Divide({0},{1})", n1, n2);
            Console.WriteLine("Return: {0}", result);
            return result;
        }
    }
}

```

```

using System;
using System.ServiceModel;
using System.ServiceModel.Description;
using GettingStartedLib;

namespace GettingStartedHost
{
    class Program
    {
        static void Main(string[] args)
        {
            // Step 1 of the address configuration procedure: Create a URI to serve as the base address.
            Uri baseAddress = new Uri("http://localhost:8000/ServiceModelSamples/Service");

            // Step 2 of the hosting procedure: Create ServiceHost
            ServiceHost selfHost = new ServiceHost(typeof(CalculatorService), baseAddress);

            try
            {
                // Step 3 of the hosting procedure: Add a service endpoint.
                selfHost.AddServiceEndpoint(typeof(ICalculator), new WSHttpBinding(), "CalculatorService");

                // Step 4 of the hosting procedure: Enable metadata exchange.
                ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
                smb.HttpGetEnabled = true;
                selfHost.Description.Behaviors.Add(smb);

                // Step 5 of the hosting procedure: Start (and then stop) the service.
                selfHost.Open();
                Console.WriteLine("The service is ready.");
                Console.WriteLine("Press <ENTER> to terminate service.");
                Console.WriteLine();
                Console.ReadLine();

                // Close the ServiceHostBase to shutdown the service.
                selfHost.Close();
            }
            catch (CommunicationException ce)
            {
                Console.WriteLine("An exception occurred: {0}", ce.Message);
                selfHost.Abort();
            }
        }
    }
}

```

Imports System.ServiceModel

Namespace GettingStartedLib

```

<ServiceContract(Namespace:="http://Microsoft.ServiceModel.Samples")> _
Public Interface ICalculator

```

```

    <OperationContract(>> _
    Function Add(ByVal n1 As Double, ByVal n2 As Double) As Double
    <OperationContract(>> _
    Function Subtract(ByVal n1 As Double, ByVal n2 As Double) As Double
    <OperationContract(>> _
    Function Multiply(ByVal n1 As Double, ByVal n2 As Double) As Double
    <OperationContract(>> _
    Function Divide(ByVal n1 As Double, ByVal n2 As Double) As Double

```

End Interface

End Namespace

```
Imports System.ServiceModel
```

```
Namespace GettingStartedLib
```

```
    Public Class CalculatorService
```

```
        Implements ICalculator
```

```
        Public Function Add(ByVal n1 As Double, ByVal n2 As Double) As Double Implements ICalculator.Add
```

```
            Dim result As Double = n1 + n2
```

```
            ' Code added to write output to the console window.
```

```
            Console.WriteLine("Received Add({0},{1})", n1, n2)
```

```
            Console.WriteLine("Return: {0}", result)
```

```
            Return result
```

```
        End Function
```

```
        Public Function Subtract(ByVal n1 As Double, ByVal n2 As Double) As Double Implements  
ICalculator.Subtract
```

```
            Dim result As Double = n1 - n2
```

```
            Console.WriteLine("Received Subtract({0},{1})", n1, n2)
```

```
            Console.WriteLine("Return: {0}", result)
```

```
            Return result
```

```
        End Function
```

```
        Public Function Multiply(ByVal n1 As Double, ByVal n2 As Double) As Double Implements  
ICalculator.Multiply
```

```
            Dim result As Double = n1 * n2
```

```
            Console.WriteLine("Received Multiply({0},{1})", n1, n2)
```

```
            Console.WriteLine("Return: {0}", result)
```

```
            Return result
```

```
        End Function
```

```
        Public Function Divide(ByVal n1 As Double, ByVal n2 As Double) As Double Implements ICalculator.Divide
```

```
            Dim result As Double = n1 / n2
```

```
            Console.WriteLine("Received Divide({0},{1})", n1, n2)
```

```
            Console.WriteLine("Return: {0}", result)
```

```
            Return result
```

```
        End Function
```

```
    End Class
```

```
End Namespace
```

```
Imports System.ServiceModel
Imports System.ServiceModel.Description
Imports GettingStartedLibVB.GettingStartedLib

Module Service

    Class Program
        Shared Sub Main()
            ' Step 1 of the address configuration procedure: Create a URI to serve as the base address.
            Dim baseAddress As New Uri("http://localhost:8000/ServiceModelSamples/Service")

            ' Step 2 of the hosting procedure: Create ServiceHost
            Dim selfHost As New ServiceHost(GetType(CalculatorService), baseAddress)
            Try

                ' Step 3 of the hosting procedure: Add a service endpoint.
                ' Add a service endpoint
                selfHost.AddServiceEndpoint( _
                    GetType(ICalculator), _
                    New WSHttpBinding(), _
                    "CalculatorService")

                ' Step 4 of the hosting procedure: Enable metadata exchange.
                ' Enable metadata exchange
                Dim smb As New ServiceMetadataBehavior()
                smb.HttpGetEnabled = True
                selfHost.Description.Behaviors.Add(smb)

                ' Step 5 of the hosting procedure: Start (and then stop) the service.
                selfHost.Open()
                Console.WriteLine("The service is ready.")
                Console.WriteLine("Press <ENTER> to terminate service.")
                Console.WriteLine()
                Console.ReadLine()

                ' Close the ServiceHostBase to shutdown the service.
                selfHost.Close()
            Catch ce As CommunicationException
                Console.WriteLine("An exception occurred: {0}", ce.Message)
                selfHost.Abort()
            End Try
        End Sub
    End Class

End Module
```

NOTE

Services such as this one require permission to register HTTP addresses on the machine for listening. Administrator accounts have this permission, but non-administrator accounts must be granted permission for HTTP namespaces. For more information about how to configure namespace reservations, see [Configuring HTTP and HTTPS](#). When running under Visual Studio, the service.exe must be run with administrator privileges.

Next steps

Now the service is running. In the next task, you create a WCF client.

[How to: Create a WCF client](#)

For troubleshooting information, see [Troubleshooting the Getting Started Tutorial](#).

See also

- [Getting Started](#)
- [Self-Host](#)

How to: Create a Windows Communication Foundation Client

10/24/2018 • 3 minutes to read • [Edit Online](#)

This is the fourth of six tasks required to create a Windows Communication Foundation (WCF) application. For an overview of all six of the tasks, see the [Getting Started Tutorial](#) topic.

This topic describes how to retrieve metadata from a WCF service and use it to create a WCF proxy that can access the service. This task is completed by using the **Add Service Reference** functionality provided by Visual Studio. This tool obtains the metadata from the service's MEX endpoint and generates a managed source code file for a client proxy in the language you have chosen (C# by default). In addition to creating the client proxy, the tool also creates or updates the client configuration file which enables the client application to connect to the service at one of its endpoints.

NOTE

You can also use the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) tool to generate the proxy class and configuration instead of using **Add Service Reference** in Visual Studio.

NOTE

When calling a WCF service from a class library project in Visual Studio, you can use the **Add Service Reference** feature to automatically generate a proxy and associated configuration file. The configuration file will not be used by the class library project. You need to add the settings in the generated configuration file to the app.config file for the executable that calls the class library.

The client application uses the generated proxy class to communicate with the service. This procedure is described in [How to: Use a Client](#).

To create a Windows Communication Foundation client

1. Create a new console application project in Visual Studio. Right-click on the Getting Started solution in **Solution Explorer** and select **Add > New Project**. In the **Add New Project** dialog, on the left-hand side, select the **Windows Desktop** category under **Visual C#** or **Visual Basic**. Select the **Console App (.NET Framework)** template, and then name the project **GettingStartedClient**.
2. Add a reference to System.ServiceModel to the GettingStartedClient project. Right-click on the **References** folder under the GettingStartedClient project in **Solution Explorer**, and then select **Add Reference**. In the **Add Reference** dialog, select **Framework** on the left-hand side of the dialog under **Assemblies**. Find and select **System.ServiceModel**, and then choose **OK**. Save the solution by selecting **File > Save All**.
3. Add a service reference to the Calculator Service.
 - a. First, start up the GettingStartedHost console application.
 - b. Once the host is running, right-click the **References** folder under the GettingStartedClient project in **Solution Explorer** and select **Add > Service Reference**.
 - c. Enter the following URL in the address box of the **Add Service Reference** dialog:

<http://localhost:8000/GettingStarted/CalculatorService>

d. Choose **Go**.

The CalculatorService is displayed in the **Services** list box. Double-click CalculatorService to expand it and show the service contracts implemented by the service. Leave the default namespace as-is and choose **OK**.

When you add a reference to a service using Visual Studio, a new item appears in **Solution Explorer** under the **Service References** folder under the GettingStartedClient project. If you use the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) tool, a source code file and app.config file are generated.

You can also use the command-line tool [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) with the appropriate switches to create the client code. The following example generates a code file and a configuration file for the service. The first example shows how to generate the proxy in VB, and the second shows how to generate the proxy in C#:

```
svcutil.exe /language:vb /out:generatedProxy.vb /config:app.config  
http://localhost:8000/GettingStarted/CalculatorService
```

```
svcutil.exe /language:cs /out:generatedProxy.cs /config:app.config  
http://localhost:8000/GettingStarted/CalculatorService
```

Next steps

You've created the proxy that the client application will use to call the calculator service. Proceed to the next topic in the series.

[How to: Configure a Client](#)

See also

- [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#)
- [Getting Started](#)
- [Self-Host](#)
- [How to: Publish Metadata for a Service Using a Configuration File](#)
- [How to: Use Svcutil.exe to Download Metadata Documents](#)

How to: Configure a Basic Windows Communication Foundation Client

9/18/2018 • 2 minutes to read • [Edit Online](#)

This is the fifth of six tasks required to create a basic Windows Communication Foundation (WCF) application. For an overview of all six of the tasks, see the [Getting Started Tutorial](#) topic.

This topic discusses the client configuration file that was generated using the **Add Service Reference** functionality of Visual Studio or the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#). Configuring the client consists of specifying the endpoint that the client uses to access the service. An endpoint has an address, a binding, and a contract, and each of these must be specified in the process of configuring the client.

Configure a Windows Communication Foundation client

Open the generated configuration file (App.config) from the GettingStartedClient project. The following example is a view of the generated configuration file. Under the `<system.serviceModel>` section, find the `<endpoint>` element.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <!-- specifies the version of WCF to use-->
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5,Profile=Client" />
  </startup>
  <system.serviceModel>
    <bindings>
      <!-- Uses wsHttpBinding-->
      <wsHttpBinding>
        <binding name="WSHttpBinding_ICalculator" />
      </wsHttpBinding>
    </bindings>
    <client>
      <!-- specifies the endpoint to use when calling the service -->
      <endpoint address="http://localhost:8000/ServiceModelSamples/Service/CalculatorService"
        binding="wsHttpBinding" bindingConfiguration="WSHttpBinding_ICalculator"
        contract="ServiceReference1.ICalculator" name="WSHttpBinding_ICalculator">
        <identity>
          <userPrincipalName value="migree@redmond.corp.microsoft.com" />
        </identity>
      </endpoint>
    </client>
  </system.serviceModel>
</configuration>
```

This example configures the endpoint that the client uses to access the service that is located at the following address: `http://localhost:8000/ServiceModelSamples/Service/CalculatorService`.

The endpoint element specifies that the `ServiceReference1.ICalculator` service contract is used for communication between the WCF client and service. The WCF channel is configured with the system-provided [WSHttpBinding](#). This contract was generated by using **Add Service Reference** in Visual Studio. It is essentially a copy of the contract that was defined in the GettingStartedLib project. The [WSHttpBinding](#) binding specifies HTTP as the transport, interoperable security, and other configuration details.

For more information about how to use the generated client with this configuration, see [How to: Use a Client](#).

Next steps

[How to: Use a WCF client](#)

See also

- [Using Bindings to Configure Services and Clients](#)
- [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#)
- [How to: Create a Client](#)
- [Getting Started](#)
- [Self-Host](#)

How to: Use a Windows Communication Foundation Client

9/18/2018 • 3 minutes to read • [Edit Online](#)

This is the last of six tasks required to create a basic Windows Communication Foundation (WCF) application. For an overview of all six of the tasks, see the [Getting Started Tutorial](#) topic.

Once a Windows Communication Foundation (WCF) proxy has been created and configured, a client instance can be created and the client application can be compiled and used to communicate with the WCF service. This topic describes procedures for instantiating and using a WCF client. This procedure does three things:

1. Instantiates a WCF client.
2. Calls the service operations from the generated proxy.
3. Closes the client once the operation call is completed.

Use a Windows Communication Foundation client

Open the Program.cs or Program.vb file from the GettingStartedClient project and replace the existing code with the following code:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using GettingStartedClient.ServiceReference1;

namespace GettingStartedClient
{
    class Program
    {
        static void Main(string[] args)
        {
            //Step 1: Create an instance of the WCF proxy.
            CalculatorClient client = new CalculatorClient();

            // Step 2: Call the service operations.
            // Call the Add service operation.
            double value1 = 100.00D;
            double value2 = 15.99D;
            double result = client.Add(value1, value2);
            Console.WriteLine("Add({0},{1}) = {2}", value1, value2, result);

            // Call the Subtract service operation.
            value1 = 145.00D;
            value2 = 76.54D;
            result = client.Subtract(value1, value2);
            Console.WriteLine("Subtract({0},{1}) = {2}", value1, value2, result);

            // Call the Multiply service operation.
            value1 = 9.00D;
            value2 = 81.25D;
            result = client.Multiply(value1, value2);
            Console.WriteLine("Multiply({0},{1}) = {2}", value1, value2, result);

            // Call the Divide service operation.
            value1 = 22.00D;
            value2 = 7.00D;
            result = client.Divide(value1, value2);
            Console.WriteLine("Divide({0},{1}) = {2}", value1, value2, result);

            //Step 3: Closing the client gracefully closes the connection and cleans up resources.
            client.Close();
        }
    }
}

```

```
Imports System
Imports System.Collections.Generic
Imports System.Text
Imports System.ServiceModel
Imports GettingStartedClientVB2.ServiceReference1

Module Module1

    Sub Main()
        ' Step 1: Create an instance of the WCF proxy
        Dim Client As New CalculatorClient()

        'Step 2: Call the service operations.
        'Call the Add service operation.
        Dim value1 As Double = 100D
        Dim value2 As Double = 15.99D
        Dim result As Double = Client.Add(value1, value2)
        Console.WriteLine("Add({0},{1}) = {2}", value1, value2, result)

        'Call the Subtract service operation.
        value1 = 145D
        value2 = 76.54D
        result = Client.Subtract(value1, value2)
        Console.WriteLine("Subtract({0},{1}) = {2}", value1, value2, result)

        'Call the Multiply service operation.
        value1 = 9D
        value2 = 81.25D
        result = Client.Multiply(value1, value2)
        Console.WriteLine("Multiply({0},{1}) = {2}", value1, value2, result)

        'Call the Divide service operation.
        value1 = 22D
        value2 = 7D
        result = Client.Divide(value1, value2)
        Console.WriteLine("Divide({0},{1}) = {2}", value1, value2, result)

        ' Step 3: Closing the client gracefully closes the connection and cleans up resources.
        Client.Close()

        Console.WriteLine()
        Console.WriteLine("Press <ENTER> to terminate client.")
        Console.ReadLine()

    End Sub

End Module
```

Notice the `using` or `Imports` statement that imports the `GettingStartedClient.ServiceReference1`. This imports the code generated by **Add Service Reference** in Visual Studio. The code instantiates the WCF proxy and then calls each of the service operations exposed by the calculator service, closes the proxy, and terminates.

You have now completed the tutorial. You defined a service contract, implemented the service contract, generated a WCF proxy, configured a WCF client application, and then used the proxy to call service operations. To test out the application, first run `GettingStartedHost` to start the service and then run `GettingStartedClient`.

The output from `GettingStartedHost` should look like this:

```
The service is ready.Press <ENTER> to terminate service.Received Add(100,15.99)Return: 115.99Received
Subtract(145,76.54)Return: 68.46Received Multiply(9,81.25)Return: 731.25Received Divide(22,7)Return:
3.14285714285714
```

The output from `GettingStartedClient` should look like this:


```
Add(100,15.99) = 115.99Subtract(145,76.54) = 68.46Multiply(9,81.25) = 731.25Divide(22,7) =  
3.14285714285714Press <ENTER> to terminate client.
```

See also

- [Building Clients](#)
- [How to: Create a Client](#)
- [Getting Started Tutorial](#)
- [Basic WCF Programming](#)
- [How to: Create a Duplex Contract](#)
- [How to: Access Services with a Duplex Contract](#)
- [Getting Started](#)
- [Self-Host](#)

Troubleshooting the Getting Started Tutorial

8/7/2018 • 2 minutes to read • [Edit Online](#)

This topic lists the most common problems encountered when working through the Getting Started Tutorial and how to resolve them.

I am unable to find the project files on my hard drive.

Visual Studio saves project files in `c:\users\\Documents\<Visual Studio version>\Projects`.

Attempting to run the service application: HTTP could not register URL

`http://+:8000/ServiceModelSamples/Service/`. **Your process does not have access rights to this namespace.**

The process that hosts a WCF service must be run with Administrative privileges. If you are running the service from inside Visual Studio, you must run Visual Studio as an Administrator. To do so click **Start**, right-click **Visual Studio <version>** and select **Run As Administrator**. If you are running the service from a command-line prompt in a console window, you must start the console window as an Administrator in a similar way. Click **Start**, right-click **Command Prompt** and select **Run As Administrator**.

Attempting to use the Svcutil.exe tool: 'svcutil' is not recognized as an internal or external command, operable program or batch file.

Svcutil.exe must be in the system path. The easiest solution is to use the Command Prompt. Click **Start**, select **All Programs, Visual Studio <version>, Visual Studio Tools**, and **Developer Command Prompt for Visual Studio**. This command prompt sets the system path to the correct locations for all tools shipped as part of Visual Studio.

Unable to find the App.config file generated by Svcutil.exe.

The **Add Existing Item** dialog only displays files with the following extensions by default: .cs, .resx, .settings, .xsd, .wsdl. You can specify that you want to see all file types by selecting **All Files (*.*)** in the drop-down list box in the lower right corner of the **Add Existing Item** dialog box.

Compiling the client application: 'CalculatorClient' does not contain a definition for '<method name>' and no extension method '<method name>' accepting a first argument of type 'CalculatorClient' could be found (are you missing a using directive or an assembly reference?)

Only those methods that are marked with the `ServiceOperationAttribute` are exposed to the outside world. If you omitted the `ServiceOperationAttribute` attribute from one of the methods in the `ICalculator` interface, you get this error message when compiling a client application that makes a call to the operation missing the attribute.

Compiling the client application: The type or namespace name 'CalculatorClient' could not be found (are you missing a using directive or an assembly reference?)

You get this error if you do not add the Proxy.cs or Proxy.vb file to your client project.

Running the client: Unhandled Exception: System.ServiceModel.EndpointNotFoundException: Could not connect to `http://localhost:8000/ServiceModelSamples/Service/CalculatorService`. TCP error code 10061: No connection could be made because the target machine actively refused it.

This error occurs if you run the client application without running the service.

Unhandled Exception: System.ServiceModel.Security.SecurityNegotiationException: SOAP security negotiation with `http://localhost:8000/ServiceModelSamples/Service/CalculatorService` for target

`http://localhost:8000/ServiceModelSamples/Service/CalculatorService` failed

This error occurs on a domain-joined computer that does not have network connectivity. Either connect your computer to the network or turn off security for both the client and the service. For the service, modify the code that creates the WSHttpBinding to the following.

```
// Step 3 of the hosting procedure: Add a service endpoint  
selfhost.AddServiceEndpoint(typeof(ICalculator), new WSHttpBinding(SecurityMode.None), "CalculatorService");
```

For the client, change the **<security>** element under the **<binding>** element to the following:

```
<security mode="None" />
```

See Also

[Getting Started Tutorial](#)

[WCF Troubleshooting Quickstart](#)

[Troubleshooting Setup Issues](#)

Basic WCF Programming

8/31/2018 • 2 minutes to read • [Edit Online](#)

This section presents the fundamentals for creating Windows Communication Foundation (WCF) applications.

In This Section

[Basic Programming Lifecycle](#)

Describes the lifecycle of designing, building, and deploying WCF service and client applications.

[Designing and Implementing Services](#)

Describes how to design and implement a service contract, choose a message exchange pattern, specify a fault contract, and other basic aspects of services.

[Configuring Services](#)

Describes how to configure a WCF service to support the contract requirements, customize local runtime behavior, and indicate the address to publish the service.

[Hosting Services](#)

Describes the basics of hosting services in an application.

[Building Clients](#)

Describes how to obtain metadata from services, convert that into WCF client code, handle security issues, and build, configure, and host an WCF client.

[Introduction to Extensibility](#)

Describes how to extend WCF to create custom solutions.

[WCF Troubleshooting Quickstart](#)

Describes some of the most common issues that occur, what you can do to solve them, and where to locate more information about the issue.

[WCF and ASP.NET Web API](#)

Discusses the two technologies, how they relate to each other, and when to use them.

Reference

[System.ServiceModel](#)

[System.ServiceModel.Channels](#)

[System.ServiceModel.Description](#)

Related Sections

[System Requirements](#)

[Conceptual Overview](#)

[Getting Started Tutorial](#)

[Guidelines and Best Practices](#)

[Windows Communication Foundation Tools](#)

Windows Communication Foundation Samples

Getting Started

IIS Hosting Using Inline Code

Self-Host

Basic Programming Lifecycle

5/5/2018 • 2 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) enables applications to communicate whether they are on the same computer, across the Internet, or on different application platforms. This topic outlines the tasks that are required to build a WCF application. For a working sample application, see [Getting Started Tutorial](#).

The Basic Tasks

The basic tasks to perform are, in order:

1. Define the service contract. A service contract specifies the signature of a service, the data it exchanges, and other contractually required data. For more information, see [Designing Service Contracts](#).
2. Implement the contract. To implement a service contract, create a class that implements the contract and specify custom behaviors that the runtime should have. For more information, see [Implementing Service Contracts](#).
3. Configure the service by specifying endpoints and other behavior information. For more information, see [Configuring Services](#).
4. Host the service. For more information, see [Hosting Services](#).
5. Build a client application. For more information, see [Building Clients](#).

Although the topics in this section follow this order, some scenarios do not start at the beginning. For example, if you want to build a client for a pre-existing service, you start at step 5. Or if you are building a service that others will use, you may skip step 5.

Once you are familiar with developing service contracts, you can also read [Introduction to Extensibility](#). If you have problems with your service, check [WCF Troubleshooting Quickstart](#) to see whether others have the same or similar problems.

See Also

[Implementing Service Contracts](#)

Designing and Implementing Services

8/31/2018 • 5 minutes to read • [Edit Online](#)

This section shows you how to define and implement WCF contracts. A service contract specifies what an endpoint communicates to the outside world. At a more concrete level, it is a statement about a set of specific messages organized into basic message exchange patterns (MEPs), such as request/reply, one-way, and duplex. If a service contract is a logically related set of message exchanges, a service operation is a single message exchange. For example, a `Hello` operation must obviously accept one message (so the caller can announce the greeting) and may or may not return a message (depending upon the courtesy of the operation).

For more information about contracts and other core Windows Communication Foundation (WCF) concepts, see [Fundamental Windows Communication Foundation Concepts](#). This topic focuses on understanding service contracts. For more information about how to build clients that use service contracts to connect to services, see [WCF Client Overview](#).

Overview

This topic provides a high level conceptual orientation to designing and implementing WCF services. Subtopics provide more detailed information about the specifics of design and implementation. Before designing and implementing your WCF application, it is recommended that you:

- Understand what a service contract is, how it works, and how to create one.
- Understand that contracts state minimum requirements that runtime configuration or the hosting environment may not support.

Service Contracts

A service contract specifies the following:

- The operations a contract exposes.
- The signature of the operations in terms of messages exchanged.
- The data types of these messages.
- The location of the operations.
- The specific protocols and serialization formats that are used to support successful communication with the service.

For example, a purchase order contract might have a `CreateOrder` operation that accepts an input of order information types and returns success or failure information, including an order identifier. It might also have a `GetOrderStatus` operation that accepts an order identifier and returns order status information. A service contract of this sort would specify:

1. That the purchase order contract consisted of `CreateOrder` and `GetOrderStatus` operations.
2. That the operations have specified input messages and output messages.
3. The data that these messages can carry.
4. Categorical statements about the communication infrastructure necessary to successfully process the messages. For example, these details include whether and what forms of security are required to establish

successful communication.

To convey this kind of information to other applications on many platforms (including non-Microsoft platforms), XML service contracts are publicly expressed in standard XML formats, such as [Web Services Description Language](#) (WSDL) and [XML Schema](#) (XSD), among others. Developers for many platforms can use this public contract information to create applications that can communicate with the service, both because they understand the language of the specification and because those languages are designed to enable interoperation by describing the public forms, formats, and protocols that the service supports. For more information about how WCF handles this kind of information, see [Metadata](#).

Contracts can be expressed many ways, and while WSDL and XSD are excellent languages to describe services in an accessible way, they are difficult languages to use directly and are merely descriptions of a service, not service contract implementations. Therefore, WCF applications use managed attributes, interfaces, and classes both to define the structure of a service and to implement it.

The resulting contract defined in managed types can be *exported* as metadata—WSDL and XSD—when needed by clients or other service implementers. The result is a straightforward programming model that can be described (using public metadata) to any client application. The details of the underlying SOAP messages, the transportation and security-related information, and so on, can be left to WCF, which performs the necessary conversions to and from the service contract type system to the XML type system automatically.

For more information about designing contracts, see [Designing Service Contracts](#). For more information about implementing contracts, see [Implementing Service Contracts](#).

Messages Up Front and Center

Using managed interfaces, classes, and methods to model service operations is straightforward when you are used to remote procedure call (RPC)-style method signatures, in which passing parameters into a method and receiving return values is the normal form of requesting functionality from an object or other type of code. For example, programmers using managed languages such as Visual Basic and C++ COM can apply their knowledge of the RPC-style approach (whether using objects or interfaces) to the creation of WCF service contracts without experiencing the problems inherent in RPC-style distributed object systems. Service orientation provides the benefits of loosely coupled, message-oriented programming while retaining the ease and familiarity of the RPC programming experience.

Many programmers are more comfortable with message-oriented application programming interfaces, such as message queues like Microsoft MSMQ, the [System.Messaging](#) namespaces in the .NET Framework, or sending unstructured XML in HTTP requests, to name a few. For more information about programming at the message level, see [Using Message Contracts](#), [Service Channel-Level Programming](#), and [Interoperability with POX Applications](#).

Understanding the Hierarchy of Requirements

A service contract groups the operations; specifies the message exchange pattern, message types, and data types those messages carry; and indicates categories of run-time behavior an implementation must have to support the contract (for example, it may require that messages be encrypted and signed). The service contract itself does not specify precisely how these requirements are met, only that they must be. The type of encryption or the manner in which a message is signed is up to the implementation and configuration of a compliant service.

Notice the way that the contract requires certain things of the service contract implementation and the run-time configuration to add behavior. The set of requirements that must be met to expose a service for use builds on the preceding set of requirements. If a contract makes requirements of the implementation, an implementation can require yet more of the configuration and bindings that enable the service to run. Finally, the host application must also support any requirements that the service configuration and bindings add.

This additive requirement process is important to keep in mind while designing, implementing, configuring, and hosting a Windows Communication Foundation (WCF) service application. For example, the contract can specify

that it needs to support a session. If so, then you must configure the binding to support that contractual requirement, or the service implementation will not work. Or if your service requires Windows Integrated Authentication and is hosted in Internet Information Services (IIS), the Web application in which the service resides must have Windows Integrated Authentication turned on and anonymous support turned off. For more information about the features and impact of the different service host application types, see [Hosting Services](#).

See Also

[Designing Service Contracts](#)

[Implementing Service Contracts](#)

Designing Service Contracts

5/5/2018 • 15 minutes to read • [Edit Online](#)

This topic describes what service contracts are, how they are defined, what operations are available (and the implications for the underlying message exchanges), what data types are used, and other issues that help you design operations that satisfy the requirements of your scenario.

Creating a Service Contract

Services expose a number of operations. In Windows Communication Foundation (WCF) applications, define the operations by creating a method and marking it with the [OperationContractAttribute](#) attribute. Then, to create a service contract, group together your operations, either by declaring them within an interface marked with the [ServiceContractAttribute](#) attribute, or by defining them in a class marked with the same attribute. (For a basic example, see [How to: Define a Service Contract](#).)

Any methods that do not have a [OperationContractAttribute](#) attribute are not service operations and are not exposed by WCF services.

This topic describes the following decision points when designing a service contract:

- Whether to use classes or interfaces.
- How to specify the data types you want to exchange.
- The types of exchange patterns you can use.
- Whether you can make explicit security requirements part of the contract.
- The restrictions for operation inputs and outputs.

Classes or Interfaces

Both classes and interfaces represent a grouping of functionality and, therefore, both can be used to define a WCF service contract. However, it is recommended that you use interfaces because they directly model service contracts. Without an implementation, interfaces do no more than define a grouping of methods with certain signatures. Implement a service contract interface and you have implemented a WCF service.

All the benefits of managed interfaces apply to service contract interfaces:

- Service contract interfaces can extend any number of other service contract interfaces.
- A single class can implement any number of service contracts by implementing those service contract interfaces.
- You can modify the implementation of a service contract by changing the interface implementation, while the service contract remains the same.
- You can version your service by implementing the old interface and the new one. Old clients connect to the original version, while newer clients can connect to the newer version.

NOTE

When inheriting from other service contract interfaces, you cannot override operation properties, such as the name or namespace. If you attempt to do so, you create a new operation in the current service contract.

For an example of using an interface to create a service contract, see [How to: Create a Service with a Contract Interface](#).

You can, however, use a class to define a service contract and implement that contract at the same time. The advantage of creating your services by applying [ServiceContractAttribute](#) and [OperationContractAttribute](#) directly to the class and the methods on the class, respectively, is speed and simplicity. The disadvantages are that managed classes do not support multiple inheritance, and as a result they can only implement one service contract at a time. In addition, any modification to the class or method signatures modifies the public contract for that service, which can prevent unmodified clients from using your service. For more information, see [Implementing Service Contracts](#).

For an example that uses a class to create a service contract and implements it at the same time, see [How to: Create a Service with a Contract Class](#).

At this point, you should understand the difference between defining your service contract by using an interface and by using a class. The next step is deciding what data can be passed back and forth between a service and its clients.

Parameters and Return Values

Each operation has a return value and a parameter, even if these are `void`. However, unlike a local method, in which you can pass references to objects from one object to another, service operations do not pass references to objects. Instead, they pass copies of the objects.

This is significant because each type used in a parameter or return value must be serializable; that is, it must be possible to convert an object of that type into a stream of bytes and from a stream of bytes into an object.

Primitive types are serializable by default, as are many types in the .NET Framework.

NOTE

The value of the parameter names in the operation signature are part of the contract and are case sensitive. If you want to use the same parameter name locally but modify the name in the published metadata, see the [System.ServiceModel.MessageParameterAttribute](#).

Data Contracts

Service-oriented applications like Windows Communication Foundation (WCF) applications are designed to interoperate with the widest possible number of client applications on both Microsoft and non-Microsoft platforms. For the widest possible interoperability, it is recommended that you mark your types with the [DataContractAttribute](#) and [DataMemberAttribute](#) attributes to create a data contract, which is the portion of the service contract that describes the data that your service operations exchange.

Data contracts are opt-in style contracts: No type or data member is serialized unless you explicitly apply the data contract attribute. Data contracts are unrelated to the access scope of the managed code: Private data members can be serialized and sent elsewhere to be accessed publicly. (For a basic example of a data contract, see [How to: Create a Basic Data Contract for a Class or Structure](#).) WCF handles the definition of the underlying SOAP messages that enable the operation's functionality as well as the serialization of your data types into and out of the body of the messages. As long as your data types are serializable, you do not need to think about the underlying message exchange infrastructure when designing your operations.

Although the typical WCF application uses the [DataContractAttribute](#) and [DataMemberAttribute](#) attributes to create data contracts for operations, you can use other serialization mechanisms. The standard [ISerializable](#), [SerializableAttribute](#) and [IXmlSerializable](#) mechanisms all work to handle the serialization of your data types into the underlying SOAP messages that carry them from one application to another. You can employ more serialization strategies if your data types require special support. For more information about the choices for

serialization of data types in WCF applications, see [Specifying Data Transfer in Service Contracts](#).

Mapping Parameters and Return Values to Message Exchanges

Service operations are supported by an underlying exchange of SOAP messages that transfer application data back and forth, in addition to the data required by the application to support certain standard security, transaction, and session-related features. Because this is the case, the signature of a service operation dictates a certain underlying *message exchange pattern* (MEP) that can support the data transfer and the features an operation requires. You can specify three patterns in the WCF programming model: request/reply, one-way, and duplex message patterns.

Request/Reply

A request/reply pattern is one in which a request sender (a client application) receives a reply with which the request is correlated. This is the default MEP because it supports an operation in which one or more parameters are passed to the operation and a return value is passed back to the caller. For example, the following C# code example shows a basic service operation that takes one string and returns a string.

```
[OperationContractAttribute]
string Hello(string greeting);
```

The following is the equivalent Visual Basic code.

```
<OperationContractAttribute()>
Function Hello (ByVal greeting As String) As String
```

This operation signature dictates the form of underlying message exchange. If no correlation existed, WCF cannot determine for which operation the return value is intended.

Note that unless you specify a different underlying message pattern, even service operations that return `void` (`Nothing` in Visual Basic) are request/reply message exchanges. The result for your operation is that unless a client invokes the operation asynchronously, the client stops processing until the return message is received, even though that message is empty in the normal case. The following C# code example shows an operation that does not return until the client has received an empty message in response.

```
[OperationContractAttribute]
void Hello(string greeting);
```

The following is the equivalent Visual Basic code.

```
<OperationContractAttribute()>
Sub Hello (ByVal greeting As String)
```

The preceding example can slow client performance and responsiveness if the operation takes a long time to perform, but there are advantages to request/reply operations even when they return `void`. The most obvious one is that SOAP faults can be returned in the response message, which indicates that some service-related error condition has occurred, whether in communication or processing. SOAP faults that are specified in a service contract are passed to the client application as a [FaultException<TDetail>](#) object, where the type parameter is the type specified in the service contract. This makes notifying clients about error conditions in WCF services easy. For more information about exceptions, SOAP faults, and error handling, see [Specifying and Handling Faults in Contracts and Services](#). To see an example of a request/reply service and client, see [How to: Create a Request-Reply Contract](#). For more information about issues with the request-reply pattern, see [Request-Reply Services](#).

One-way

If the client of a WCF service application should not wait for the operation to complete and does not process SOAP faults, the operation can specify a one-way message pattern. A one-way operation is one in which a client

invokes an operation and continues processing after WCF writes the message to the network. Typically this means that unless the data being sent in the outbound message is extremely large the client continues running almost immediately (unless there is an error sending the data). This type of message exchange pattern supports event-like behavior from a client to a service application.

A message exchange in which one message is sent and none are received cannot support a service operation that specifies a return value other than `void`; in this case an [InvalidOperationException](#) exception is thrown.

No return message also means that there can be no SOAP fault returned to indicate any errors in processing or communication. (Communicating error information when operations are one-way operations requires a duplex message exchange pattern.)

To specify a one-way message exchange for an operation that returns `void`, set the [IsOneWay](#) property to `true`, as in the following C# code example.

```
[OperationContractAttribute(IsOneWay=true)]  
void Hello(string greeting);
```

The following is the equivalent Visual Basic code.

```
<OperationContractAttribute(IsOneWay := True)>  
Sub Hello (ByVal greeting As String)
```

This method is identical to the preceding request/reply example, but setting the [IsOneWay](#) property to `true` means that although the method is identical, the service operation does not send a return message and clients return immediately once the outbound message has been handed to the channel layer. For an example, see [How to: Create a One-Way Contract](#). For more information about the one-way pattern, see [One-Way Services](#).

Duplex

A duplex pattern is characterized by the ability of both the service and the client to send messages to each other independently whether using one-way or request/reply messaging. This form of two-way communication is useful for services that must communicate directly to the client or for providing an asynchronous experience to either side of a message exchange, including event-like behavior.

The duplex pattern is slightly more complex than the request/reply or one-way patterns because of the additional mechanism for communicating with the client.

To design a duplex contract, you must also design a callback contract and assign the type of that callback contract to the [CallbackContract](#) property of the [ServiceContractAttribute](#) attribute that marks your service contract.

To implement a duplex pattern, you must create a second interface that contains the method declarations that are called on the client.

For an example of creating a service, and a client that accesses that service, see [How to: Create a Duplex Contract](#) and [How to: Access Services with a Duplex Contract](#). For a working sample, see [Duplex](#). For more information about issues using duplex contracts, see [Duplex Services](#).

Caution

When a service receives a duplex message, it looks at the `ReplyTo` element in that incoming message to determine where to send the reply. If the channel that is used to receive the message is not secured, then an untrusted client could send a malicious message with a target machine's `ReplyTo`, leading to a denial of service (DOS) of that target machine.

Out and Ref Parameters

In most cases, you can use `in` parameters (`ByVal` in Visual Basic) and `out` and `ref` parameters (`ByRef` in Visual Basic). Because both `out` and `ref` parameters indicate that data is returned from an operation, an operation signature such as the following specifies that a request/reply operation is required even though the

operation signature returns `void`.

```
[ServiceContractAttribute]
public interface IMyContract
{
    [OperationContractAttribute]
    public void PopulateData(ref CustomDataType data);
}
```

The following is the equivalent Visual Basic code.

```
<ServiceContractAttribute()> _
Public Interface IMyContract
    <OperationContractAttribute()> _
    Public Sub PopulateData(ByRef data As CustomDataType)
End Interface
```

The only exceptions are those cases in which your signature has a particular structure. For example, you can use the [NetMsmqBinding](#) binding to communicate with clients only if the method used to declare an operation returns `void`; there can be no output value, whether it is a return value, `ref`, or `out` parameter.

In addition, using `out` or `ref` parameters requires that the operation have an underlying response message to carry back the modified object. If your operation is a one-way operation, an [InvalidOperationException](#) exception is thrown at runtime.

Specify Message Protection Level on the Contract

When designing your contract, you must also decide the message protection level of services that implement your contract. This is necessary only if message security is applied to the binding in the contract's endpoint. If the binding has security turned off (that is, if the system-provided binding sets the [System.ServiceModel.SecurityMode](#) to the value [SecurityMode.None](#)) then you do not have to decide on the message protection level for the contract. In most cases, system-provided bindings with message-level security applied provide a sufficient protection level and you do not have to consider the protection level for each operation or for each message.

The protection level is a value that specifies whether the messages (or message parts) that support a service are signed, signed and encrypted, or sent without signatures or encryption. The protection level can be set at various scopes: At the service level, for a particular operation, for a message within that operation, or a message part. Values set at one scope become the default value for smaller scopes unless explicitly overridden. If a binding configuration cannot provide the required minimum protection level for the contract, an exception is thrown. And when no protection level values are explicitly set on the contract, the binding configuration controls the protection level for all messages if the binding has message security. This is the default behavior.

IMPORTANT

Deciding whether to explicitly set various scopes of a contract to less than the full protection level of [ProtectionLevel.EncryptAndSign](#) is generally a decision that trades some degree of security for increased performance. In these cases, your decisions must revolve around your operations and the value of the data they exchange. For more information, see [Securing Services](#).

For example, the following code example does not set either the [ProtectionLevel](#) or the [ProtectionLevel](#) property on the contract.

```
[ServiceContract]
public interface ISampleService
{
    [OperationContractAttribute]
    public string GetString();

    [OperationContractAttribute]
    public int GetInt();
}
```

The following is the equivalent Visual Basic code.

```
<ServiceContractAttribute()> _
Public Interface ISampleService

    <OperationContractAttribute()> _
    Public Function GetString()As String

    <OperationContractAttribute()> _
    Public Function GetData() As Integer

End Interface
```

When interacting with an `ISampleService` implementation in an endpoint with a default [WSHttpBinding](#) (the default [System.ServiceModel.SecurityMode](#), which is [Message](#)), all messages are encrypted and signed because this is the default protection level. However, when an `ISampleService` service is used with a default [BasicHttpBinding](#) (the default [SecurityMode](#), which is [None](#)), all messages are sent as text because there is no security for this binding and so the protection level is ignored (that is, the messages are neither encrypted nor signed). If the [SecurityMode](#) was changed to [Message](#), then these messages would be encrypted and signed (because that would now be the binding's default protection level).

If you want to explicitly specify or adjust the protection requirements for your contract, set the [ProtectionLevel](#) property (or any of the [ProtectionLevel](#) properties at a smaller scope) to the level your service contract requires. In this case, using an explicit setting requires the binding to support that setting at a minimum for the scope used. For example, the following code example specifies one [ProtectionLevel](#) value explicitly, for the `GetGuid` operation.

```
[ServiceContract]
public interface IExplicitProtectionLevelSampleService
{
    [OperationContractAttribute]
    public string GetString();

    [OperationContractAttribute(ProtectionLevel=ProtectionLevel.None)]
    public int GetInt();
    [OperationContractAttribute(ProtectionLevel=ProtectionLevel.EncryptAndSign)]
    public int GetGuid();
}
```

The following is the equivalent Visual Basic code.

```

<ServiceContract(> _
Public Interface IExplicitProtectionLevelSampleService
    <OperationContract(> _
        Public Function GetString() As String
        End Function

    <OperationContract(ProtectionLevel := ProtectionLevel.None)> _
        Public Function GetInt() As Integer
        End Function

    <OperationContractAttribute(ProtectionLevel := ProtectionLevel.EncryptAndSign)> _
        Public Function GetGuid() As Integer
        End Function

End Interface

```

A service that implements this `IExplicitProtectionLevelSampleService` contract and has an endpoint that uses the default [WSHttpBinding](#) (the default [System.ServiceModel.SecurityMode](#), which is [Message](#)) has the following behavior:

- The `GetString` operation messages are encrypted and signed.
- The `GetInt` operation messages are sent as unencrypted and unsigned (that is, plain) text.
- The `GetGuid` operation [System.Guid](#) is returned in a message that is encrypted and signed.

For more information about protection levels and how to use them, see [Understanding Protection Level](#). For more information about security, see [Securing Services](#).

Other Operation Signature Requirements

Some application features require a particular kind of operation signature. For example, the [NetMsmqBinding](#) binding supports durable services and clients, in which an application can restart in the middle of communication and pick up where it left off without missing any messages. (For more information, see [Queues in WCF](#).) However, durable operations must take only one `in` parameter and have no return value.

Another example is the use of [Stream](#) types in operations. Because the [Stream](#) parameter includes the entire message body, if an input or an output (that is, `ref` parameter, `out` parameter, or return value) is of type [Stream](#), then it must be the only input or output specified in your operation. In addition, the parameter or return type must be either [Stream](#), [System.ServiceModel.Channels.Message](#), or [System.Xml.Serialization.IXmlSerializable](#). For more information about streams, see [Large Data and Streaming](#).

Names, Namespaces, and Obfuscation

The names and namespaces of the .NET types in the definition of contracts and operations are significant when contracts are converted into WSDL and when contract messages are created and sent. Therefore, it is strongly recommended that service contract names and namespaces are explicitly set using the `Name` and `Namespace` properties of all supporting contract attributes such as the [ServiceContractAttribute](#), [OperationContractAttribute](#), [DataContractAttribute](#), [DataMemberAttribute](#), and other contract attributes.

One result of this is that if the names and namespaces are not explicitly set, the use of IL obfuscation on the assembly alters the contract type names and namespaces and results in modified WSDL and wire exchanges that typically fail. If you do not set the contract names and namespaces explicitly but do intend to use obfuscation, use the [ObfuscationAttribute](#) and [ObfuscateAssemblyAttribute](#) attributes to prevent the modification of the contract type names and namespaces.

See Also

[How to: Create a Request-Reply Contract](#)
[How to: Create a One-Way Contract](#)

[How to: Create a Duplex Contract](#)

[Specifying Data Transfer in Service Contracts](#)

[Specifying and Handling Faults in Contracts and Services](#)

[Using Sessions](#)

[Synchronous and Asynchronous Operations](#)

[Reliable Services](#)

[Services and Transactions](#)

Specifying and Handling Faults in Contracts and Services

5/5/2018 • 5 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) applications handle error situations by mapping managed exception objects to SOAP fault objects and SOAP fault objects to managed exception objects. The topics in this section discuss how to design contracts to expose error conditions as custom SOAP faults, how to return such faults as part of service implementation, and how clients catch such faults.

Error Handling Overview

In all managed applications, processing errors are represented by [Exception](#) objects. In SOAP-based applications such as WCF applications, service methods communicate processing error information using SOAP fault messages. SOAP faults are message types that are included in the metadata for a service operation and therefore create a fault contract that clients can use to make their operation more robust or interactive. In addition, because SOAP faults are expressed to clients in XML form, it is a highly interoperable type system that clients on any SOAP platform can use, increasing the reach of your WCF application.

Because WCF applications run under both types of error systems, any managed exception information that is sent to the client must be converted from exceptions into SOAP faults on the service, sent, and converted from SOAP faults to fault exceptions in WCF clients. In the case of duplex clients, client contracts can also send SOAP faults back to a service. In either case, you can use the default service exception behaviors, or you can explicitly control whether—and how—exceptions are mapped to fault messages.

Two types of SOAP faults can be sent: *declared* and *undeclared*. Declared SOAP faults are those in which an operation has a [System.ServiceModel.FaultContractAttribute](#) attribute that specifies a custom SOAP fault type. *Undeclared* SOAP faults are not specified in the contract for an operation.

It is strongly recommended that service operations declare their faults by using the [FaultContractAttribute](#) attribute to formally specify all SOAP faults that a client can expect to receive in the normal course of an operation. It is also recommended that you return in a SOAP fault only the information that a client must know to minimize information disclosure.

Typically, services (and duplex clients) take the following steps to successfully integrate error handling into their applications:

- Map exception conditions to custom SOAP faults.
- Clients and services send and receive SOAP faults as exceptions.

In addition, WCF clients and services can use undeclared soap faults for debugging purposes and can extend the default error behavior. The following sections discuss these tasks and concepts.

Map Exceptions to SOAP Faults

The first step in creating an operation that handles error conditions is to decide under what conditions a client application should be informed about errors. Some operations have error conditions specific to their functionality. For example, a `PurchaseOrder` operation might return specific information to customers who are no longer permitted to initiate a purchase order. In other cases, such as a `Calculator` service, a more general `MathFault` SOAP fault may be able to describe all error conditions across an entire service. Once the error conditions of clients of your service are identified, a custom SOAP fault can be constructed and the operation can

be marked as returning that SOAP fault when its corresponding error condition arises.

For more information about this step of developing your service or client, see [Defining and Specifying Faults](#).

Clients and Services Handle SOAP Faults as Exceptions

Identifying operation error conditions, defining custom SOAP faults, and marking those operations as returning those faults are the first steps in successful error handling in WCF applications. The next step is to properly implement the sending and receiving of these faults. Typically services send faults to inform client applications about error conditions, but duplex clients can also send SOAP faults to services.

For more information, see [Sending and Receiving Faults](#).

Undeclared SOAP Faults and Debugging

Declared SOAP faults are extremely useful for building robust, interoperable, distributed applications. However, in some cases it is useful for a service (or duplex client) to send an undeclared SOAP fault, one that is not mentioned in the Web Services Description Language (WSDL) for that operation. For example, when developing a service, unexpected situations can occur in which it is useful for debugging purposes to send information back to the client. In addition, you can set the [ServiceBehaviorAttribute.IncludeExceptionDetailInFaults](#) property or the [ServiceDebugBehavior.IncludeExceptionDetailInFaults](#) property to `true` to permit WCF clients to obtain information about internal service operation exceptions. Both sending individual faults and setting the debugging behavior properties are described in [Sending and Receiving Faults](#).

IMPORTANT

Because managed exceptions can expose internal application information, setting [ServiceBehaviorAttribute.IncludeExceptionDetailInFaults](#) or [ServiceDebugBehavior.IncludeExceptionDetailInFaults](#) to `true` can permit WCF clients to obtain information about internal service operation exceptions, including personally identifiable or other sensitive information.

Therefore, setting [ServiceBehaviorAttribute.IncludeExceptionDetailInFaults](#) or [ServiceDebugBehavior.IncludeExceptionDetailInFaults](#) to `true` is recommended only as a way to temporarily debug a service application. In addition, the WSDL for a method that returns unhandled managed exceptions in this way does not contain the contract for the [FaultException<TDetail>](#) of type [ExceptionDetail](#). Clients must expect the possibility of an unknown SOAP fault (returned to WCF clients as [System.ServiceModel.FaultException](#) objects) to obtain the debugging information properly.

Customizing Error Handling with IErrorHandler

If you have special requirements to either customize the response message to the client when an application-level exception happens or perform some custom processing after the response message is returned, implement the [System.ServiceModel.Dispatcher.IErrorHandler](#) interface.

Fault Serialization Issues

When deserializing a fault contract, WCF first attempts to match the fault contract name in the SOAP message with the fault contract type. If it cannot find an exact match it will then search the list of available fault contracts in alphabetical order for a compatible type. If two fault contracts are compatible types (one is a subclass of another, for example) the wrong type may be used to de-serialize the fault. This only occurs if the fault contract does not specify a name, namespace, and action. To prevent this issue from occurring, always fully qualify fault contracts by specifying the name, namespace, and action attributes. Additionally if you have defined a number of related fault contracts derived from a shared base class, make sure to mark any new members with

`[DataMember(IsRequired=true)]`. For more information on this `IsRequired` attribute see, [DataMemberAttribute](#).

This will prevent a base class from being a compatible type and force the fault to be deserialized into the correct derived type.

See Also

[FaultException](#)

[FaultContractAttribute](#)

[FaultException](#)

[XmlSerializer](#)

[XmlSerializerFormatAttribute](#)

[FaultContractAttribute](#)

[CommunicationException](#)

[Action](#)

[Code](#)

[Reason](#)

[SubCode](#)

[IsOneWay](#)

[Defining and Specifying Faults](#)

Defining and Specifying Faults

5/5/2018 • 5 minutes to read • [Edit Online](#)

SOAP faults convey error condition information from a service to a client and, in the duplex case, from a client to a service in an interoperable way. This topic discusses when and how to define custom fault content and specify which operations can return them. For more information about how a service, or duplex client, can send those faults and how a client or service application handles these faults, see [Sending and Receiving Faults](#). For an overview of error handling in Windows Communication Foundation (WCF) applications, see [Specifying and Handling Faults in Contracts and Services](#).

Overview

Declared SOAP faults are those in which an operation has a [System.ServiceModel.FaultContractAttribute](#) that specifies a custom SOAP fault type. Undeclared SOAP faults are those that are not specified in the contract for an operation. This topic helps you identify those error conditions and create a fault contract for your service that clients can use to properly handle those error conditions when notified by custom SOAP faults. The basic tasks are, in order:

1. Define the error conditions that a client of your service should know about.
2. Define the custom content of the SOAP faults for those error conditions.
3. Mark your operations so that the specific SOAP faults that they throw are exposed to clients in WSDL.

Defining Error Conditions That Clients Should Know About

SOAP faults are publicly described messages that carry fault information for a particular operation. Because they are described along with other operation messages in WSDL, clients know and, therefore, expect to handle such faults when invoking an operation. But because WCF services are written in managed code, deciding which error conditions in managed code are to be converted into faults and returned to the client provides you the opportunity to separate error conditions and bugs in your service from the formal error conversation you have with a client.

For example, the following code example shows an operation that takes two integers and returns another integer. Several exceptions can be thrown here, so when designing the fault contract, you must determine which error conditions are important for your client. In this case, the service should detect the [System.DivideByZeroException](#) exception.

```
[ServiceContract]
public class CalculatorService
{
    [OperationContract]
    int Divide(int a, int b)
    {
        if (b==0) throw new Exception("Division by zero!");
        return a/b;
    }
}
```

```

<ServiceContract> _
Public Class CalculatorService
    <OperationContract> _
    Public Function Divide(ByVal a As Integer, ByVal b As Integer) _
        As Integer
        If (b==0) Then
            Throw New Exception("Division by zero!")
        Return a/b
    End Function
End Class

```

In the preceding example the operation can either return a custom SOAP fault that is specific to dividing by zero, a custom fault that is specific to math operations but that contains information specific to dividing by zero, multiple faults for several different error situations, or no SOAP fault at all.

Define the Content of Error Conditions

Once an error condition has been identified as one that can usefully return a custom SOAP fault, the next step is to define the contents of that fault and ensure that the content structure can be serialized. The code example in the preceding section shows an error specific to a `Divide` operation, but if there are other operations on the `Calculator` service, then a single custom SOAP fault can inform the client of all calculator error conditions, `Divide` included. The following code example shows the creation of a custom SOAP fault, `MathFault`, which can report errors made using all math operations, including `Divide`. While the class can specify an operation (the `Operation` property) and a value that describes the problem (the `ProblemType` property), the class and these properties must be serializable to be transferred to the client in a custom SOAP fault. Therefore, the [System.Runtime.Serialization.DataContractAttribute](#) and [System.Runtime.Serialization.DataMemberAttribute](#) attributes are used to make the type and its properties serializable and as interoperable as possible.

```

// Define a math fault data contract
[DataContract(Namespace="http://Microsoft.ServiceModel.Samples")]
public class MathFault
{
    private string operation;
    private string problemType;

    [DataMember]
    public string Operation
    {
        get { return operation; }
        set { operation = value; }
    }

    [DataMember]
    public string ProblemType
    {
        get { return problemType; }
        set { problemType = value; }
    }
}

```

```

' Define a math fault data contract
<DataContract([Namespace]:="http://Microsoft.ServiceModel.Samples")> _
Public Class MathFault

    Private m_operation As String
    Private m_problemType As String

    <DataMember()> _
    Public Property Operation() As String

        Get

            Return m_operation

        End Get

        Set(ByVal value As String)

            m_operation = value

        End Set

    End Property

    <DataMember()> _
    Public Property ProblemType() As String

        Get

            Return m_problemType

        End Get

        Set(ByVal value As String)

            m_problemType = value

        End Set

    End Property

End Class

```

For more information about how to ensure your data is serializable, see [Specifying Data Transfer in Service Contracts](#). For a list of the serialization support that [System.Runtime.Serialization.DataContractSerializer](#) provides, see [Types Supported by the Data Contract Serializer](#).

Mark Operations to Establish the Fault Contract

Once a serializable data structure that is returned as part of a custom SOAP fault is defined, the last step is to mark your operation contract as throwing a SOAP fault of that type. To do this, use the [System.ServiceModel.FaultContractAttribute](#) attribute and pass the type of the custom data type that you have constructed. The following code example shows how to use the [FaultContractAttribute](#) attribute to specify that the `Divide` operation can return a SOAP fault of type `MathFault`. Other math-based operations can now also specify that they can return a `MathFault`.

```

[OperationContract]
[FaultContract(typeof(MathFault))]
int Divide(int n1, int n2);

```

```
<OperationContract(> _  
<FaultContract(GetType(MathFault))> _  
Function Divide(ByVal n1 As Integer, ByVal n2 As Integer) As Integer
```

An operation can specify that it returns more than one custom fault by marking that operation with more than one [FaultContractAttribute](#) attribute.

The next step, to implement the fault contract in your operation implementation, is described in the topic [Sending and Receiving Faults](#).

SOAP, WSDL, and Interoperability Considerations

In some circumstances, especially when interoperating with other platforms, it may be important to control the way a fault appears in a SOAP message or the way it is described in the WSDL metadata.

The [FaultContractAttribute](#) attribute has a [Name](#) property that allows control of the WSDL fault element name that is generated in the metadata for that fault.

According to the SOAP standard, a fault can have an [Action](#), a [Code](#), and a [Reason](#). The [Action](#) is controlled by the [Action](#) property. The [Code](#) property and [Reason](#) property are both properties of the [System.ServiceModel.FaultException](#) class, which is the parent class of the generic [System.ServiceModel.FaultException<TDetail>](#). The [Code](#) property includes a [SubCode](#) member.

When accessing non-services that generate faults, certain limitations exist. WCF supports only faults with detail types that the schema describes and that are compatible with data contracts. For example, as mentioned above, WCF does not support faults that use XML attributes in their detail types, or faults with more than one top-level element in the detail section.

See Also

[FaultContractAttribute](#)

[DataContractAttribute](#)

[DataMemberAttribute](#)

[Specifying and Handling Faults in Contracts and Services](#)

[Sending and Receiving Faults](#)

[How to: Declare Faults in Service Contracts](#)

[Understanding Protection Level](#)

[How to: Set the ProtectionLevel Property](#)

[Specifying Data Transfer in Service Contracts](#)

How to: Declare Faults in Service Contracts

5/4/2018 • 3 minutes to read • [Edit Online](#)

In managed code, exceptions are thrown when error conditions occur. In Windows Communication Foundation (WCF) applications, however, service contracts specify what error information is returned to clients by declaring SOAP faults in the service contract. For an overview of the relationship between exceptions and faults, see [Specifying and Handling Faults in Contracts and Services](#).

Create a service contract that specifies a SOAP fault

1. Create a service contract that contains at least one operation. For an example, see [How to: Define a Service Contract](#).
2. Select an operation that can specify an error condition about which clients can expect to be notified. To decide which error conditions justify returning SOAP faults to clients, see [Specifying and Handling Faults in Contracts and Services](#).
3. Apply a [System.ServiceModel.FaultContractAttribute](#) to the selected operation and pass a serializable fault type to the constructor. For details about creating and using serializable types, see [Specifying Data Transfer in Service Contracts](#). The following example shows how to specify that the `SampleMethod` operation can result in a `GreetingFault`.

```
[OperationContract]
[FaultContractAttribute(
    typeof(GreetingFault),
    Action="http://www.contoso.com/GreetingFault",
    ProtectionLevel=ProtectionLevel.EncryptAndSign
)]
string SampleMethod(string msg);
```

```
<OperationContract, FaultContractAttribute(GetType(GreetingFault),
Action:= "http://www.contoso.com/GreetingFault", ProtectionLevel:=ProtectionLevel.EncryptAndSign)> _
Function SampleMethod(ByVal msg As String) As String
```

4. Repeat steps 2 and 3 for all operations in the contract that communicate error conditions to clients.

Implementing an Operation to Return a Specified SOAP Fault

Once an operation has specified that a specific SOAP fault can be returned (such as in the preceding procedure) to communicate an error condition to a calling application, the next step is to implement that specification.

Throw the specified SOAP fault in the operation

1. When a [FaultContractAttribute](#)-specified error condition occurs in an operation, throw a new [System.ServiceModel.FaultException<TDetail>](#) where the specified SOAP fault is the type parameter. The following example shows how to throw the `GreetingFault` in the `SampleMethod` shown in the preceding procedure and in the following Code section.

```
throw new FaultException<GreetingFault>(new GreetingFault("A Greeting error occurred. You said: " +
msg));
```

```
Throw New FaultException(Of GreetingFault)(New GreetingFault("A Greeting error occurred. You said: " &
msg))
End If
```

Example

The following code example shows an implementation of a single operation that specifies a `GreetingFault` for the `SampleMethod` operation.

```

using System;
using System.Collections.Generic;
using System.Net.Security;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace Microsoft.WCF.Documentation
{
    [ServiceContract(Namespace="http://microsoft.wcf.documentation")]
    public interface ISampleService{
        [OperationContract]
        [FaultContractAttribute(
            typeof(GreetingFault),
            Action="http://www.contoso.com/GreetingFault",
            ProtectionLevel=ProtectionLevel.EncryptAndSign
        )]
        string SampleMethod(string msg);
    }

    [DataContractAttribute]
    public class GreetingFault
    {
        private string report;

        public GreetingFault(string message)
        {
            this.report = message;
        }

        [DataMemberAttribute]
        public string Message
        {
            get { return this.report; }
            set { this.report = value; }
        }
    }

    class SampleService : ISampleService
    {
        #region ISampleService Members

        public string SampleMethod(string msg)
        {
            Console.WriteLine("Client said: " + msg);
            // Generate intermittent error behavior.
            Random rnd = new Random(DateTime.Now.Millisecond);
            int test = rnd.Next(5);
            if (test % 2 != 0)
                return "The service greets you: " + msg;
            else
                throw new FaultException<GreetingFault>(new GreetingFault("A Greeting error occurred. You said: " +
msg));
        }

        #endregion
    }
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.Net.Security
Imports System.Runtime.Serialization
Imports System.ServiceModel
Imports System.Text

Namespace Microsoft.WCF.Documentation
    <ServiceContract(Namespace:="http://microsoft.wcf.documentation")> _
    Public Interface ISampleService
        <OperationContract, FaultContractAttribute(GetType(GreetingFault),
Action:="http://www.contoso.com/GreetingFault", ProtectionLevel:=ProtectionLevel.EncryptAndSign)> _
        Function SampleMethod(ByVal msg As String) As String
    End Interface

    <DataContractAttribute> _
    Public Class GreetingFault
        Private report As String

        Public Sub New(ByVal message As String)
            Me.report = message
        End Sub

        <DataMemberAttribute> _
        Public Property Message() As String
            Get
                Return Me.report
            End Get
            Set(ByVal value As String)
                Me.report = value
            End Set
        End Property
    End Class

    Friend Class SampleService
        Implements ISampleService
        #Region "ISampleService Members"

        Public Function SampleMethod(ByVal msg As String) As String Implements ISampleService.SampleMethod
            Console.WriteLine("Client said: " & msg)
            ' Generate intermittent error behavior.
            Dim rand As New Random(DateTime.Now.Millisecond)
            Dim test As Integer = rand.Next(5)
            If test Mod 2 <> 0 Then
                Return "The service greets you: " & msg
            Else
                Throw New FaultException(Of GreetingFault)(New GreetingFault("A Greeting error occurred. You said: " &
msg))
            End If
        End Function

        #End Region
    End Class
End Namespace

```

See Also

[System.ServiceModel.FaultContractAttribute](#)
[System.ServiceModel.FaultException<TDetail>](#)

Sending and Receiving Faults

11/13/2018 • 7 minutes to read • [Edit Online](#)

SOAP faults convey error condition information from a service to a client and in the duplex case from a client to a service in an interoperable way. Typically a service defines custom fault content and specifies which operations can return them. (For more information, see [Defining and Specifying Faults](#).) This topic discusses how a service or duplex client can send those faults when the corresponding error condition has occurred and how a client or service application handles these faults. For an overview of error handling in Windows Communication Foundation (WCF) applications, see [Specifying and Handling Faults in Contracts and Services](#).

Sending SOAP Faults

Declared SOAP faults are those in which an operation has a [System.ServiceModel.FaultContractAttribute](#) that specifies a custom SOAP fault type. Undeclared SOAP faults are those that are not specified in the contract for an operation.

Sending Declared Faults

To send a declared SOAP fault, detect the error condition for which the SOAP fault is appropriate and throw a new [System.ServiceModel.FaultException<TDetail>](#) where the type parameter is a new object of the type specified in the [FaultContractAttribute](#) for that operation. The following code example shows the use of [FaultContractAttribute](#) to specify that the `SampleMethod` operation can return a SOAP fault with the detail type of `GreetingFault`.

```
[OperationContract]
[FaultContractAttribute(
    typeof(GreetingFault),
    Action="http://www.contoso.com/GreetingFault",
    ProtectionLevel=ProtectionLevel.EncryptAndSign
)]
string SampleMethod(string msg);
```

```
<OperationContract, FaultContractAttribute(GetType(GreetingFault),
Action="http://www.contoso.com/GreetingFault", ProtectionLevel:=ProtectionLevel.EncryptAndSign)> _
Function SampleMethod(ByVal msg As String) As String
```

To convey the `GreetingFault` error information to the client, catch the appropriate error condition and throw a new [System.ServiceModel.FaultException<TDetail>](#) of type `GreetingFault` with a new `GreetingFault` object as the argument, as in the following code example. If the client is an WCF client application, it experiences this as a managed exception where the type is [System.ServiceModel.FaultException<TDetail>](#) of type `GreetingFault`.

```
throw new FaultException<GreetingFault>(new GreetingFault("A Greeting error occurred. You said: " + msg));
```

```
Throw New FaultException(Of GreetingFault)(New GreetingFault("A Greeting error occurred. You said: " &
msg))
End If
```

Sending Undeclared Faults

Sending undeclared faults can be very useful to quickly diagnose and debug problems in WCF applications, but

its usefulness as a debugging tool is limited. More generally, when debugging it is recommended that you use the [ServiceDebugBehavior.IncludeExceptionDetailInFaults](#) property. When you set this value to true, clients experience such faults as [FaultException<TDetail>](#) exceptions of type [ExceptionDetail](#).

IMPORTANT

Because managed exceptions can expose internal application information, setting [ServiceBehaviorAttribute.IncludeExceptionDetailInFaults](#) or [ServiceDebugBehavior.IncludeExceptionDetailInFaults](#) to `true` can permit WCF clients to obtain information about internal service operation exceptions, including personally identifiable or other sensitive information.

Therefore, setting [ServiceBehaviorAttribute.IncludeExceptionDetailInFaults](#) or [ServiceDebugBehavior.IncludeExceptionDetailInFaults](#) to `true` is only recommended as a way of temporarily debugging a service application. In addition, the WSDL for a method that returns unhandled managed exceptions in this way does not contain the contract for the [FaultException<TDetail>](#) of type [ExceptionDetail](#). Clients must expect the possibility of an unknown SOAP fault (returned to WCF clients as [System.ServiceModel.FaultException](#) objects) to obtain the debugging information properly.

To send an undeclared SOAP fault, throw a [System.ServiceModel.FaultException](#) object (that is, not the generic type [FaultException<TDetail>](#)) and pass the string to the constructor. This is exposed to the WCF client applications as a thrown [System.ServiceModel.FaultException](#) exception where the string is available by calling the [FaultException<TDetail>.ToString](#) method.

NOTE

If you declare a SOAP fault of type string, and then throw this in your service as a [FaultException<TDetail>](#) where the type parameter is a [System.String](#) the string value is assigned to the [FaultException<TDetail>.Detail](#) property, and is not available from [FaultException<TDetail>.ToString](#).

Handling Faults

In WCF clients, SOAP faults that occur during communication that are of interest to client applications are raised as managed exceptions. While there are many exceptions that can occur during the execution of any program, applications using the WCF client programming model can expect to handle exceptions of the following two types as a result of communication.

- [TimeoutException](#)
- [CommunicationException](#)

[TimeoutException](#) objects are thrown when an operation exceeds the specified timeout period.

[CommunicationException](#) objects are thrown when there is some recoverable communication error condition on either the service or the client.

The [CommunicationException](#) class has two important derived types, [FaultException](#) and the generic [FaultException<TDetail>](#) type.

[FaultException](#) exceptions are thrown when a listener receives a fault that is not expected or specified in the operation contract; usually this occurs when the application is being debugged and the service has the [ServiceDebugBehavior.IncludeExceptionDetailInFaults](#) property set to `true`.

[FaultException<TDetail>](#) exceptions are thrown on the client when a fault that is specified in the operation contract is received in response to a two-way operation (that is, a method with an [OperationContractAttribute](#) attribute with [IsOneWay](#) set to `false`).

NOTE

When an WCF service has the [ServiceBehaviorAttribute.IncludeExceptionDetailInFaults](#) or [ServiceDebugBehavior.IncludeExceptionDetailInFaults](#) property set to `true` the client experiences this as an undeclared [FaultException<TDetail>](#) of type [ExceptionDetail](#). Clients can either catch this specific fault or handle the fault in a catch block for [FaultException](#).

Typically, only [FaultException<TDetail>](#), [TimeoutException](#), and [CommunicationException](#) exceptions are of interest to clients and services.

NOTE

Other exceptions, of course, do occur. Unexpected exceptions include catastrophic failures like [System.OutOfMemoryException](#); typically applications should not catch such methods.

Catch Fault Exceptions in the Correct Order

Because [FaultException<TDetail>](#) derives from [FaultException](#), and [FaultException](#) derives from [CommunicationException](#), it is important to catch these exceptions in the proper order. If, for example, you have a try/catch block in which you first catch [CommunicationException](#), all specified and unspecified SOAP faults are handled there; any subsequent catch blocks to handle a custom [FaultException<TDetail>](#) exception are never invoked.

Remember that one operation can return any number of specified faults. Each fault is a unique type and must be handled separately.

Handle Exceptions When Closing the Channel

Most of the preceding discussion has to do with faults sent in the course of processing application messages, that is, messages explicitly sent by the client when the client application calls operations on the WCF client object.

Even with local objects disposing the object can either raise or mask exceptions that occur during the recycling process. Something similar can occur when you use WCF client objects. When you call operations you are sending messages over an established connection. Closing the channel can throw exceptions if the connection cannot be cleanly closed or is already closed, even if all the operations returned properly.

Typically, client object channels are closed in one of the following ways:

- When the WCF client object is recycled.
- When the client application calls [ClientBase<TChannel>.Close](#).
- When the client application calls [ICommunicationObject.Close](#).
- When the client application calls an operation that is a terminating operation for a session.

In all cases, closing the channel instructs the channel to begin closing any underlying channels that may be sending messages to support complex functionality at the application level. For example, when a contract requires sessions a binding attempts to establish a session by exchanging messages with the service channel until a session is established. When the channel is closed, the underlying session channel notifies the service that the session is terminated. In this case, if the channel has already aborted, closed, or is otherwise unusable (for example, when a network cable is unplugged), the client channel cannot inform the service channel that the session is terminated and an exception can result.

Abort the Channel If Necessary

Because closing the channel can also throw exceptions, then, it is recommended that in addition to catching fault exceptions in the correct order, it is important to abort the channel that was used in making the call in the catch

block.

If the fault conveys error information specific to an operation and it remains possible that others can use it, there is no need to abort the channel (although these cases are rare). In all other cases, it is recommended that you abort the channel. For a sample that demonstrates all of these points, see [Expected Exceptions](#).

The following code example shows how to handle SOAP fault exceptions in a basic client application, including a declared fault and an undeclared fault.

NOTE

This sample code does not use the `using` construct. Because closing channels can throw exceptions, it is recommended that applications create a WCF client first, and then open, use, and close the WCF client in the same try block. For details, see [WCF Client Overview](#) and [Use Close and Abort to release WCF client resources](#).


```

using System;
using System.ServiceModel;
using System.ServiceModel.Channels;
using Microsoft.WCF.Documentation;

public class Client
{
    public static void Main()
    {
        // Picks up configuration from the config file.
        SampleServiceClient wcfClient = new SampleServiceClient();
        try
        {
            // Making calls.
            Console.WriteLine("Enter the greeting to send: ");
            string greeting = Console.ReadLine();
            Console.WriteLine("The service responded: " + wcfClient.SampleMethod(greeting));

            Console.WriteLine("Press ENTER to exit:");
            Console.ReadLine();

            // Done with service.
            wcfClient.Close();
            Console.WriteLine("Done!");
        }
        catch (TimeoutException timeProblem)
        {
            Console.WriteLine("The service operation timed out. " + timeProblem.Message);
            Console.ReadLine();
            wcfClient.Abort();
        }
        catch (FaultException<GreetingFault> greetingFault)
        {
            Console.WriteLine(greetingFault.Detail.Message);
            Console.ReadLine();
            wcfClient.Abort();
        }
        catch (FaultException unknownFault)
        {
            Console.WriteLine("An unknown exception was received. " + unknownFault.Message);
            Console.ReadLine();
            wcfClient.Abort();
        }
        catch (CommunicationException commProblem)
        {
            Console.WriteLine("There was a communication problem. " + commProblem.Message +
commProblem.StackTrace);
            Console.ReadLine();
            wcfClient.Abort();
        }
    }
}

```

```
Imports System
Imports System.ServiceModel
Imports System.ServiceModel.Channels
Imports Microsoft.WCF.Documentation

Public Class Client
    Public Shared Sub Main()
        ' Picks up configuration from the config file.
        Dim wcfClient As New SampleServiceClient()
        Try
            ' Making calls.
            Console.WriteLine("Enter the greeting to send: ")
            Dim greeting As String = Console.ReadLine()
            Console.WriteLine("The service responded: " & wcfClient.SampleMethod(greeting))

            Console.WriteLine("Press ENTER to exit:")
            Console.ReadLine()

            ' Done with service.
            wcfClient.Close()
            Console.WriteLine("Done!")
        Catch timeProblem As TimeoutException
            Console.WriteLine("The service operation timed out. " & timeProblem.Message)
            Console.ReadLine()
            wcfClient.Abort()
        Catch greetingFault As FaultException(Of GreetingFault)
            Console.WriteLine(greetingFault.Detail.Message)
            Console.ReadLine()
            wcfClient.Abort()
        Catch unknownFault As FaultException
            Console.WriteLine("An unknown exception was received. " & unknownFault.Message)
            Console.ReadLine()
            wcfClient.Abort()
        Catch commProblem As CommunicationException
            Console.WriteLine("There was a communication problem. " & commProblem.Message + commProblem.StackTrace)
            Console.ReadLine()
            wcfClient.Abort()
        End Try
    End Sub
End Class
```

See Also

[FaultException](#)

[FaultException<TDetail>](#)

[System.ServiceModel.CommunicationException](#)

[Expected Exceptions](#)

[Use Close and Abort to release WCF client resources](#)

Using Sessions

8/31/2018 • 10 minutes to read • [Edit Online](#)

In Windows Communication Foundation (WCF) applications, a *session* correlates a group of messages into a conversation. WCF sessions are different than the session object available in ASP.NET applications, support different behaviors, and are controlled in different ways. This topic describes the features that sessions enable in WCF applications and how to use them.

Sessions in Windows Communication Foundation Applications

When a service contract specifies that it requires a session, that contract is specifying that all calls (that is, the underlying message exchanges that support the calls) must be part of the same conversation. If a contract specifies that it allows sessions but does not require one, clients can connect and either establish a session or not establish a session. If the session ends and a message is sent through the same channel an exception is thrown.

WCF sessions have the following main conceptual features:

- They are explicitly initiated and terminated by the calling application (the WCF client).
- Messages delivered during a session are processed in the order in which they are received.
- Sessions correlate a group of messages into a conversation. Different types of correlation are possible. For instance, one session-based channel may correlate messages based on a shared network connection while another session-based channel may correlate messages based on a shared tag in the message body. The features that can be derived from the session depend on the nature of the correlation.
- There is no general data store associated with a WCF session.

If you are familiar with the [System.Web.SessionState.HttpSessionState](#) class in ASP.NET applications and the functionality it provides, you might notice the following differences between that kind of session and WCF sessions:

- ASP.NET sessions are always server-initiated.
- ASP.NET sessions are implicitly unordered.
- ASP.NET sessions provide a general data storage mechanism across requests.

This topic describes:

- The default execution behavior when using session-based bindings in the service model layer.
- The types of features that the WCF session-based, system-provided bindings provide.
- How to create a contract that declares a session requirement.
- How to understand and control the creation and termination of the session and the relationship of the session to the service instance.

Default Execution Behavior Using Sessions

A binding that attempts to initiate a session is called a *session-based* binding. Service contracts specify that they require, permit, or refuse session-based bindings by setting the [ServiceContractAttribute.SessionMode](#) property on the service contract interface (or class) to one of the [System.ServiceModel.SessionMode](#) enumeration values. By default, the value of this property is [Allowed](#), which means that if a client uses a session-based binding with a

WCF service implementation, the service establishes and uses the session provided.

When a WCF service accepts a client session, the following features are enabled by default:

1. All calls between a WCF client object are handled by the same service instance.
2. Different session-based bindings provide additional features.

System-Provided Session Types

A session-based binding supports the default association of a service instance with a particular session. However, different session-based bindings support different features in addition to enabling the session-based instancing control previously described.

WCF provides the following types of session-based application behavior:

- The [System.ServiceModel.Channels.SecurityBindingElement](#) supports security-based sessions, in which both ends of communication have agreed upon a specific secure conversation. For more information, see [Securing Services](#). For example, the [System.ServiceModel.WSHttpBinding](#) binding, which contains support for both security sessions and reliable sessions, by default uses only a secure session that encrypts and digitally signs messages.
- The [System.ServiceModel.NetTcpBinding](#) binding supports TCP/IP-based sessions to ensure that all messages are correlated by the connection at the socket level.
- The [System.ServiceModel.Channels.ReliableSessionBindingElement](#) element, which implements the WS-ReliableMessaging specification, provides support for reliable sessions in which messages can be configured to be delivered in order and exactly once, ensuring messages are received even when messages travel across multiple nodes during the conversation. For more information, see [Reliable Sessions](#).
- The [System.ServiceModel.NetMsmqBinding](#) binding provides MSMQ datagram sessions. For more information, see [Queues in WCF](#).

Setting the [SessionMode](#) property does not specify the type of session the contract requires, only that it requires one.

Creating a Contract That Requires a Session

Creating a contract that requires a session states that the group of operations that the service contract declares must all be executed within the same session and that messages must be delivered in order. To assert the level of session support that a service contract requires, set the [ServiceContractAttribute.SessionMode](#) property on your service contract interface or class to the value of the [System.ServiceModel.SessionMode](#) enumeration to specify whether the contract:

- Requires a session.
- Allows a client to establish a session.
- Prohibits a session.

Setting the [SessionMode](#) property does not, however, specify the type of session-based behavior the contract requires. It instructs WCF to confirm at runtime that the configured binding (which creates the communication channel) for the service does, does not, or can establish a session when implementing a service. Again, the binding can satisfy that requirement with any type of session-based behavior it chooses—security, transport, reliable, or some combination. The exact behavior depends on the [System.ServiceModel.SessionMode](#) value selected. If the configured binding of the service does not conform to the value of [SessionMode](#), an exception is thrown. Bindings and the channels they create that support sessions are said to be session-based.

The following service contract specifies that all operations in the `ICalculatorSession` must be exchanged within a session. None of the operations returns a value to the caller except the `Equals` method. However, the `Equals` method takes no parameters and, therefore, can only return a non-zero value inside a session in which data has already been passed to the other operations. This contract requires a session to function properly. Without a session associated with a specific client, the service instance has no way of knowing what previous data this client has sent.

```
[ServiceContract(Namespace="http://Microsoft.ServiceModel.Samples", SessionMode=SessionMode.Required)]
public interface ICalculatorSession
{
    [OperationContract(IsOneWay=true)]
    void Clear();
    [OperationContract(IsOneWay = true)]
    void AddTo(double n);
    [OperationContract(IsOneWay = true)]
    void SubtractFrom(double n);
    [OperationContract(IsOneWay = true)]
    void MultiplyBy(double n);
    [OperationContract(IsOneWay = true)]
    void DivideBy(double n);
    [OperationContract]
    double Equals();
}
```

```
<ServiceContract(Namespace="http://Microsoft.ServiceModel.Samples", SessionMode:=SessionMode.Required)> _
Public Interface ICalculatorSession

    <OperationContract(IsOneWay:=True)> _
    Sub Clear()
    <OperationContract(IsOneWay:=True)> _
    Sub AddTo(ByVal n As Double)
    <OperationContract(IsOneWay:=True)> _
    Sub SubtractFrom(ByVal n As Double)
    <OperationContract(IsOneWay:=True)> _
    Sub MultiplyBy(ByVal n As Double)
    <OperationContract(IsOneWay:=True)> _
    Sub DivideBy(ByVal n As Double)
    <OperationContract()> _
    Function Equal() As Double
End Interface
```

If a service allows a session, then a session is established and used if the client initiates one; otherwise, no session is established.

Sessions and Service Instances

If you use the default instancing behavior in WCF, all calls between a WCF client object are handled by the same service instance. Therefore, at the application level, you can think of a session as enabling application behavior similar to local call behavior. For example, when you create a local object:

- A constructor is called.
- All subsequent calls made to the WCF client object reference are processed by the same object instance.
- A destructor is called when the object reference is destroyed.

Sessions enable a similar behavior between clients and services as long as the default service instance behavior is used. If a service contract requires or supports sessions, one or more contract operations can be marked as initiating or terminating a session by setting the `IsInitiating` and `IsTerminating` properties.

Initiating operations are those that must be called as the first operation of a new session. Non-initiating operations can be called only after at least one initiating operation has been called. You can therefore create a kind of session constructor for your service by declaring initiating operations designed to take input from clients appropriate to the beginning of the service instance. (The state is associated with the session, however, and not the service object.)

Terminating operations, conversely, are those that must be called as the last message in an existing session. In the default case, WCF recycles the service object and its context after the session with which the service was associated is closed. You can, therefore, create a kind of destructor by declaring terminating operations designed to perform a function appropriate to the end of the service instance.

NOTE

Although the default behavior bears a resemblance to local constructors and destructors, it is only a resemblance. Any WCF service operation can be an initiating or terminating operation, or both at the same time. In addition, in the default case, initiating operations can be called any number of times in any order; no additional sessions are created once the session is established and associated with an instance unless you explicitly control the lifetime of the service instance (by manipulating the [System.ServiceModel.InstanceContext](#) object). Finally, the state is associated with the session and not the service object.

For example, the `ICalculatorSession` contract used in the preceding example requires that the WCF client object first call the `Clear` operation prior to any other operation and that the session with this WCF client object should terminate when it calls the `Equals` operation. The following code example shows a contract that enforces these requirements. `Clear` must be called first to initiate a session, and that session ends when `Equals` is called.

```
[ServiceContract(Namespace="http://Microsoft.ServiceModel.Samples", SessionMode=SessionMode.Required)]
public interface ICalculatorSession
{
    [OperationContract(IsOneWay=true, IsInitiating=true, IsTerminating=false)]
    void Clear();
    [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]
    void AddTo(double n);
    [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]
    void SubtractFrom(double n);
    [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]
    void MultiplyBy(double n);
    [OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]
    void DivideBy(double n);
    [OperationContract(IsInitiating = false, IsTerminating = true)]
    double Equals();
}
```

```
<ServiceContract(Namespace:= "http://Microsoft.ServiceModel.Samples", SessionMode:=SessionMode.Required)> _
Public Interface ICalculatorSession

    <OperationContract(IsOneWay:=True, IsInitiating:=True, IsTerminating:=False)> _
    Sub Clear()
    <OperationContract(IsOneWay:=True, IsInitiating:=False, IsTerminating:=False)> _
    Sub AddTo(ByVal n As Double)
    <OperationContract(IsOneWay:=True, IsInitiating:=False, IsTerminating:=False)> _
    Sub SubtractFrom(ByVal n As Double)
    <OperationContract(IsOneWay:=True, IsInitiating:=False, IsTerminating:=False)> _
    Sub MultiplyBy(ByVal n As Double)
    <OperationContract(IsOneWay:=True, IsInitiating:=False, IsTerminating:=False)> _
    Sub DivideBy(ByVal n As Double)
    <OperationContract(IsInitiating:=False, IsTerminating:=True)> _
    Function Equal() As Double
End Interface
```

Services do not start sessions with clients. In WCF client applications, a direct relationship exists between the

lifetime of the session-based channel and the lifetime of the session itself. As such, clients create new sessions by creating new session-based channels and tear down existing sessions by closing session-based channels gracefully. A client starts a session with a service endpoint by calling one of the following:

- [ICommunicationObject.Open](#) on the channel returned by a call to [ChannelFactory<TChannel>.CreateChannel](#).
- [ClientBase<TChannel>.Open](#) on the WCF client object generated by the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#).
- An initiating operation on either type of WCF client object (by default, all operations are initiating). When the first operation is called, the WCF client object automatically opens the channel and initiates a session.

Typically a client ends a session with a service endpoint by calling one of the following:

- [ICommunicationObject.Close](#) on the channel returned by a call to [ChannelFactory<TChannel>.CreateChannel](#).
- [ClientBase<TChannel>.Close](#) on the WCF client object generated by Svcutil.exe.
- A terminating operation on either type of WCF client object (by default, no operations are terminating; the contract must explicitly specify a terminating operation). When the first operation is called, the WCF client object automatically opens the channel and initiates a session.

For examples, see [How to: Create a Service That Requires Sessions](#) as well as the [Default Service Behavior](#) and [Instancing](#) samples.

For more information about clients and sessions, see [Accessing Services Using a WCF Client](#).

Sessions Interact with InstanceContext Settings

There is an interaction between the [SessionMode](#) enumeration in a contract and the [ServiceBehaviorAttribute.InstanceContextMode](#) property, which controls the association between channels and specific service objects. For more information, see [Sessions, Instancing, and Concurrency](#).

Sharing InstanceContext Objects

You can also control which session-based channel or call is associated with which [InstanceContext](#) object by performing that association yourself. For a complete example, see [InstanceContextSharing](#).

Sessions and Streaming

When you have a large amount of data to transfer, the streaming transfer mode in WCF is a feasible alternative to the default behavior of buffering and processing messages in memory in their entirety. You may get unexpected behavior when streaming calls with a session-based binding. All streaming calls are made through a single channel (the datagram channel) that does not support sessions even if the binding being used is configured to use sessions. If multiple clients make streaming calls to the same service object over a session-based binding, and the service object's concurrency mode is set to single and its instance context mode is set to `PerSession`, all calls must go through the datagram channel and so only one call is processed at a time. One or more clients may then time out. You can work around this issue by either setting the service object's `InstanceContextMode` to `PerCall` or Concurrency to multiple.

NOTE

MaxConcurrentSessions have no effect in this case because there is only one "session" available.

See Also

[IsInitiating](#)

[IsTerminating](#)

Synchronous and Asynchronous Operations

8/31/2018 • 8 minutes to read • [Edit Online](#)

This topic discusses implementing and calling asynchronous service operations.

Many applications call methods asynchronously because it enables the application to continue doing useful work while the method call runs. Windows Communication Foundation (WCF) services and clients can participate in asynchronous operation calls at two distinct levels of the application, which provide WCF applications even more flexibility to maximize throughput balanced against interactivity.

Types of Asynchronous Operations

All service contracts in WCF, no matter the parameters types and return values, use WCF attributes to specify a particular message exchange pattern between client and service. WCF automatically routes inbound and outbound messages to the appropriate service operation or running client code.

The client possesses only the service contract, which specifies the message exchange pattern for a particular operation. Clients can offer the developer any programming model they choose, so long as the underlying message exchange pattern is observed. So, too, can services implement operations in any manner, so long as the specified message pattern is observed.

The independence of the service contract from either the service or client implementation enables the following forms of asynchronous execution in WCF applications:

- Clients can invoke request/response operations asynchronously using a synchronous message exchange.
- Services can implement a request/response operation asynchronously using a synchronous message exchange.
- Message exchanges can be one-way, regardless of the implementation of the client or service.

Suggested Asynchronous Scenarios

Use an asynchronous approach in a service operation implementation if the operation service implementation makes a blocking call, such as doing I/O work. When you are in an asynchronous operation implementation, try to call asynchronous operations and methods to extend the asynchronous call path as far as possible. For example, call a `BeginOperationTwo()` from within `BeginOperationOne()`.

- Use an asynchronous approach in a client or calling application in the following cases:
- If you are invoking operations from a middle-tier application. (For more information about such scenarios, see [Middle-Tier Client Applications](#).)
- If you are invoking operations within an ASP.NET page, use asynchronous pages.
- If you are invoking operations from any application that is single threaded, such as Windows Forms or Windows Presentation Foundation (WPF). When using the event-based asynchronous calling model, the result event is raised on the UI thread, adding responsiveness to the application without requiring you to handle multiple threads yourself.
- In general, if you have a choice between a synchronous and asynchronous call, choose the asynchronous call.

Implementing an Asynchronous Service Operation

Asynchronous operations can be implemented by using one of the three following methods:

1. The task-based asynchronous pattern
2. The event-based asynchronous pattern
3. The IAsyncResult asynchronous pattern

Task-Based Asynchronous Pattern

The task-based asynchronous pattern is the preferred way to implement asynchronous operations because it is the easiest and most straight forward. To use this method simply implement your service operation and specify a return type of `Task<T>`, where T is the type returned by the logical operation. For example:

```
public class SampleService:ISampleService
{
    // ...
    public async Task<string> SampleMethodTaskAsync(string msg)
    {
        return Task<string>.Factory.StartNew(() =>
        {
            return msg;
        });
    }
    // ...
}
```

The `SampleMethodTaskAsync` operation returns `Task<string>` because the logical operation returns a string. For more information about the task-based asynchronous pattern, see [The Task-Based Asynchronous Pattern](#).

WARNING

When using the task-based asynchronous pattern, a `T:System.AggregateException` may be thrown if an exception occurs while waiting on the completion of the operation. This exception may occur on the client or services

Event-Based Asynchronous Pattern

A service that supports the Event-based Asynchronous Pattern will have one or more operations named `MethodNameAsync`. These methods may mirror synchronous versions, which perform the same operation on the current thread. The class may also have a `MethodNameCompleted` event and it may have a `MethodNameAsyncCancel` (or simply `CancelAsync`) method. A client wishing to call the operation will define an event handler to be called when the operation completes,

The following code snippet illustrates how to declare asynchronous operations using the event-based asynchronous pattern.

```

public class AsyncExample
{
    // Synchronous methods.
    public int Method1(string param);
    public void Method2(double param);

    // Asynchronous methods.
    public void Method1Async(string param);
    public void Method1Async(string param, object userState);
    public event EventHandler Method1Completed;

    public void Method2Async(double param);
    public void Method2Async(double param, object userState);
    public event EventHandler Method2Completed;

    public void CancelAsync(object userState);

    public bool IsBusy { get; }

    // Class implementation not shown.
}

```

For more information about the Event-based Asynchronous Pattern, see [The Event-Based Asynchronous Pattern](#).

IAsyncResult Asynchronous Pattern

A service operation can be implemented in an asynchronous fashion using the .NET Framework asynchronous programming pattern and marking the `<Begin>` method with the [AsyncPattern](#) property set to `true`. In this case, the asynchronous operation is exposed in metadata in the same form as a synchronous operation: It is exposed as a single operation with a request message and a correlated response message. Client programming models then have a choice. They can represent this pattern as a synchronous operation or as an asynchronous one, so long as when the service is invoked a request-response message exchange takes place.

In general, with the asynchronous nature of the systems, you should not take a dependency on the threads. The most reliable way of passing data to various stages of operation dispatch processing is to use extensions.

For an example, see [How to: Implement an Asynchronous Service Operation](#).

To define a contract operation `x` that is executed asynchronously regardless of how it is called in the client application:

- Define two methods using the pattern `BeginOperation` and `EndOperation`.
- The `BeginOperation` method includes `in` and `ref` parameters for the operation and returns an [IAsyncResult](#) type.
- The `EndOperation` method includes an [IAsyncResult](#) parameter as well as the `out` and `ref` parameters and returns the operations return type.

For example, see the following method.

```
int DoWork(string data, ref string inout, out string outonly)
```

```
Function DoWork(ByVal data As String, ByRef inout As String, _out outonly As out) As Integer
```

To create an asynchronous operation, the two methods would be:

```
[OperationContract(AsyncPattern=true)]
IAsyncResult BeginDoWork(string data,
                        ref string inout,
                        AsyncCallback callback,
                        object state);

int EndDoWork(ref string inout, out string outonly, IAsyncResult result);
```

```
<OperationContract(AsyncPattern := True)>
Function BeginDoWork(ByVal data As String, _
                    ByRef inout As String, _
                    ByVal callback As AsyncCallback, _
                    ByVal state As Object) As IAsyncResult
Function EndDoWork(ByRef inout As String, ByRef outonly As String, ByVal result As IAsyncResult) As Integer
```

NOTE

The [OperationContractAttribute](#) attribute is applied only to the `BeginDoWork` method. The resulting contract has one WSDL operation named `DoWork`.

Client-Side Asynchronous Invocations

A WCF client application can use any of three asynchronous calling models described previously

When using the task-based model, simply call the operation using the `await` keyword as shown in the following code snippet.

```
await simpleServiceClient.SampleMethodTaskAsync("hello, world");
```

Using the event-based asynchronous pattern only requires adding an event handler to receive a notification of the response -- and the resulting event is raised on the user interface thread automatically. To use this approach, specify both the `/async` and `/tcv:Version35` command options with the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#), as in the following example.

```
svcutil http://localhost:8000/servicemodelsamples/service/mex /async /tcv:Version35
```

When this is done, `Svcutil.exe` generates a WCF client class with the event infrastructure that enables the calling application to implement and assign an event handler to receive the response and take the appropriate action. For a complete example, see [How to: Call Service Operations Asynchronously](#).

The event-based asynchronous model, however, is only available in .NET Framework version 3.5. In addition, it is not supported even in .NET Framework 3.5 when a WCF client channel is created by using a `System.ServiceModel.ChannelFactory<TChannel>`. With WCF client channel objects, you must use `System.IAsyncResult` objects to invoke your operations asynchronously. To use this approach, specify the `/async` command option with the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#), as in the following example.

```
svcutil http://localhost:8000/servicemodelsamples/service/mex /async
```

This generates a service contract in which each operation is modeled as a `<Begin>` method with the `AsyncPattern` property set to `true` and a corresponding `<End>` method. For a complete example using a `ChannelFactory<TChannel>`, see [How to: Call Operations Asynchronously Using a Channel Factory](#).

In either case, applications can invoke an operation asynchronously even if the service is implemented synchronously, in the same way that an application can use the same pattern to invoke asynchronously a local

synchronous method. How the operation is implemented is not significant to the client; when the response message arrives, its content is dispatched to the client's asynchronous `< End >` method and the client retrieves the information.

One-Way Message Exchange Patterns

You can also create an asynchronous message exchange pattern in which one-way operations (operations for which the `OperationContractAttribute.IsOneWay` is `true` have no correlated response) can be sent in either direction by the client or service independently of the other side. (This uses the duplex message exchange pattern with one-way messages.) In this case, the service contract specifies a one-way message exchange that either side can implement as asynchronous calls or implementations, or not, as appropriate. Generally, when the contract is an exchange of one-way messages, the implementations can largely be asynchronous because once a message is sent the application does not wait for a reply and can continue doing other work.

Event-based Asynchronous Clients and Message Contracts

The design guidelines for the event-based asynchronous model state that if more than one value is returned, one value is returned as the `Result` property and the others are returned as properties on the `EventArgs` object. One result of this is that if a client imports metadata using the event-based asynchronous command options and the operation returns more than one value, the default `EventArgs` object returns one value as the `Result` property and the remainder are properties of the `EventArgs` object.

If you want to receive the message object as the `Result` property and have the returned values as properties on that object, use the **/messageContract** command option. This generates a signature that returns the response message as the `Result` property on the `EventArgs` object. All internal return values are then properties of the response message object.

See Also

[IsOneWay](#)

[AsyncPattern](#)

How to: Implement an Asynchronous Service Operation

8/31/2018 • 5 minutes to read • [Edit Online](#)

In Windows Communication Foundation (WCF) applications, a service operation can be implemented asynchronously or synchronously without dictating to the client how to call it. For example, asynchronous service operations can be called synchronously, and synchronous service operations can be called asynchronously. For an example that shows how to call an operation asynchronously in a client application, see [How to: Call Service Operations Asynchronously](#). For more information about synchronous and asynchronous operations, see [Designing Service Contracts](#) and [Synchronous and Asynchronous Operations](#). This topic describes the basic structure of an asynchronous service operation, the code is not complete. For a complete example of both the service and client sides see [Asynchronous](#).

Implement a service operation asynchronously

1. In your service contract, declare an asynchronous method pair according to the .NET asynchronous design guidelines. The `Begin` method takes a parameter, a callback object, and a state object, and returns a `System.IAsyncResult` and a matching `End` method that takes a `System.IAsyncResult` and returns the return value. For more information about asynchronous calls, see [Asynchronous Programming Design Patterns](#).
2. Mark the `Begin` method of the asynchronous method pair with the `System.ServiceModel.OperationContractAttribute` attribute and set the `OperationContractAttribute.AsyncPattern` property to `true`. For example, the following code performs steps 1 and 2.

```
[OperationContractAttribute(AsyncPattern=true)]
IAsyncResult BeginServiceAsyncMethod(string msg, AsyncCallback callback, object asyncState);

// Note: There is no OperationContractAttribute for the end method.
string EndServiceAsyncMethod(IAsyncResult result);
}
```

```
<OperationContractAttribute(AsyncPattern:=True)> _
Function BeginServiceAsyncMethod(ByVal msg As String, ByVal callback As AsyncCallback, ByVal
asyncState As Object) As IAsyncResult

' Note: There is no OperationContractAttribute for the end method.
Function EndServiceAsyncMethod(ByVal result As IAsyncResult) As String
End Interface
```

3. Implement the `Begin/End` method pair in your service class according to the asynchronous design guidelines. For example, the following code example shows an implementation in which a string is written to the console in both the `Begin` and `End` portions of the asynchronous service operation, and the return value of the `End` operation is returned to the client. For the complete code example, see the [Example](#) section.

```

public IAsyncResult BeginServiceAsyncMethod(string msg, AsyncCallback callback, object asyncState)
{
    Console.WriteLine("BeginServiceAsyncMethod called with: \"{0}\"", msg);
    return new CompletedAsyncResult<string>(msg);
}

public string EndServiceAsyncMethod(IAsyncResult r)
{
    CompletedAsyncResult<string> result = r as CompletedAsyncResult<string>;
    Console.WriteLine("EndServiceAsyncMethod called with: \"{0}\"", result.Data);
    return result.Data;
}

```

```

Public Function BeginServiceAsyncMethod(ByVal msg As String, ByVal callback As AsyncCallback, ByVal
asyncState As Object) As IAsyncResult Implements ISampleService.BeginServiceAsyncMethod
    Console.WriteLine("BeginServiceAsyncMethod called with: \"{0}\"", msg)
    Return New CompletedAsyncResult(Of String)(msg)
End Function

Public Function EndServiceAsyncMethod(ByVal r As IAsyncResult) As String Implements
ISampleService.EndServiceAsyncMethod
    Dim result As CompletedAsyncResult(Of String) = TryCast(r, CompletedAsyncResult(Of String))
    Console.WriteLine("EndServiceAsyncMethod called with: \"{0}\"", result.Data)
    Return result.Data
End Function

```

Example

The following code examples show:

1. A service contract interface with:
 - a. A synchronous `SampleMethod` operation.
 - b. An asynchronous `BeginSampleMethod` operation.
 - c. An asynchronous `BeginServiceAsyncMethod` / `EndServiceAsyncMethod` operation pair.
2. A service implementation using a [System.IAsyncResult](#) object.

```

using System;
using System.Collections.Generic;
using System.ServiceModel;
using System.Text;
using System.Threading;

namespace Microsoft.WCF.Documentation
{
    [ServiceContractAttribute(Namespace="http://microsoft.wcf.documentation")]
    public interface ISampleService{

        [OperationContractAttribute]
        string SampleMethod(string msg);

        [OperationContractAttribute(AsyncPattern = true)]
        IAsyncResult BeginSampleMethod(string msg, AsyncCallback callback, object asyncState);

        //Note: There is no OperationContractAttribute for the end method.
        string EndSampleMethod(IAsyncResult result);

        [OperationContractAttribute(AsyncPattern=true)]
        IAsyncResult BeginServiceAsyncMethod(string msg, AsyncCallback callback, object asyncState);
    }
}

```

```

// Note: There is no OperationContractAttribute for the end method.
string EndServiceAsyncMethod(IAsyncResult result);
}

public class SampleService : ISampleService
{
    #region ISampleService Members

    public string SampleMethod(string msg)
    {
        Console.WriteLine("Called synchronous sample method with \"{0}\"", msg);
        return "The sychronous service greets you: " + msg;
    }

    // This asynchronously implemented operation is never called because
    // there is a synchronous version of the same method.
    public IAsyncResult BeginSampleMethod(string msg, AsyncCallback callback, object asyncState)
    {
        Console.WriteLine("BeginSampleMethod called with: " + msg);
        return new CompletedAsyncResult<string>(msg);
    }

    public string EndSampleMethod(IAsyncResult r)
    {
        CompletedAsyncResult<string> result = r as CompletedAsyncResult<string>;
        Console.WriteLine("EndSampleMethod called with: " + result.Data);
        return result.Data;
    }

    public IAsyncResult BeginServiceAsyncMethod(string msg, AsyncCallback callback, object asyncState)
    {
        Console.WriteLine("BeginServiceAsyncMethod called with: \"{0}\"", msg);
        return new CompletedAsyncResult<string>(msg);
    }

    public string EndServiceAsyncMethod(IAsyncResult r)
    {
        CompletedAsyncResult<string> result = r as CompletedAsyncResult<string>;
        Console.WriteLine("EndServiceAsyncMethod called with: \"{0}\"", result.Data);
        return result.Data;
    }
    #endregion
}

// Simple async result implementation.
class CompletedAsyncResult<T> : IAsyncResult
{
    T data;

    public CompletedAsyncResult(T data)
    { this.data = data; }

    public T Data
    { get { return data; } }

    #region IAsyncResult Members
    public object AsyncState
    { get { return (object)data; } }

    public WaitHandle AsyncWaitHandle
    { get { throw new Exception("The method or operation is not implemented."); } }

    public bool CompletedSynchronously
    { get { return true; } }

    public bool IsCompleted
    { get { return true; } }
    #endregion
}

```



```
}  
}
```

```
Imports System  
Imports System.Collections.Generic  
Imports System.ServiceModel  
Imports System.Text  
Imports System.Threading  
  
Namespace Microsoft.WCF.Documentation  
    <ServiceContractAttribute(Namespace:="http://microsoft.wcf.documentation")> _  
    Public Interface ISampleService  
  
        <OperationContractAttribute> _  
        Function SampleMethod(ByVal msg As String) As String  
  
        <OperationContractAttribute(AsyncPattern := True)> _  
        Function BeginSampleMethod(ByVal msg As String, ByVal callback As AsyncCallback, ByVal asyncState As  
Object) As IAsyncResult  
  
        'Note: There is no OperationContractAttribute for the end method.  
        Function EndSampleMethod(ByVal result As IAsyncResult) As String  
  
        <OperationContractAttribute(AsyncPattern:=True)> _  
        Function BeginServiceAsyncMethod(ByVal msg As String, ByVal callback As AsyncCallback, ByVal asyncState As  
Object) As IAsyncResult  
  
        ' Note: There is no OperationContractAttribute for the end method.  
        Function EndServiceAsyncMethod(ByVal result As IAsyncResult) As String  
    End Interface  
  
    Public Class SampleService  
        Implements ISampleService  
        #Region "ISampleService Members"  
  
        Public Function SampleMethod(ByVal msg As String) As String Implements ISampleService.SampleMethod  
            Console.WriteLine("Called synchronous sample method with ""{0}""", msg)  
            Return "The synchronous service greets you: " & msg  
        End Function  
  
        ' This asynchronously implemented operation is never called because  
        ' there is a synchronous version of the same method.  
        Public Function BeginSampleMethod(ByVal msg As String, ByVal callback As AsyncCallback, ByVal asyncState  
As Object) As IAsyncResult Implements ISampleService.BeginSampleMethod  
            Console.WriteLine("BeginSampleMethod called with: " & msg)  
            Return New CompletedAsyncResult(Of String)(msg)  
        End Function  
  
        Public Function EndSampleMethod(ByVal r As IAsyncResult) As String Implements  
ISampleService.EndSampleMethod  
            Dim result As CompletedAsyncResult(Of String) = TryCast(r, CompletedAsyncResult(Of String))  
            Console.WriteLine("EndSampleMethod called with: " & result.Data)  
            Return result.Data  
        End Function  
  
        Public Function BeginServiceAsyncMethod(ByVal msg As String, ByVal callback As AsyncCallback, ByVal  
asyncState As Object) As IAsyncResult Implements ISampleService.BeginServiceAsyncMethod  
            Console.WriteLine("BeginServiceAsyncMethod called with: ""{0}""", msg)  
            Return New CompletedAsyncResult(Of String)(msg)  
        End Function  
  
        Public Function EndServiceAsyncMethod(ByVal r As IAsyncResult) As String Implements  
ISampleService.EndServiceAsyncMethod  
            Dim result As CompletedAsyncResult(Of String) = TryCast(r, CompletedAsyncResult(Of String))  
            Console.WriteLine("EndServiceAsyncMethod called with: ""{0}""", result.Data)  
            Return result.Data  
        End Function  
    End Class  
End Namespace
```

```

End Function
#End Region
End Class

' Simple async result implementation.
Friend Class CompletedAsyncResult(Of T)
    Implements IAsyncResult
    Private data_Renamed As T

    Public Sub New(ByVal data As T)
        Me.data_Renamed = data
    End Sub

    Public ReadOnly Property Data() As T
        Get
            Return data_Renamed
        End Get
    End Property

    #Region "IAsyncResult Members"
    Public ReadOnly Property AsyncState() As Object Implements IAsyncResult.AsyncState
        Get
            Return CObj(data_Renamed)
        End Get
    End Property

    Public ReadOnly Property AsyncWaitHandle() As WaitHandle Implements IAsyncResult.AsyncWaitHandle
        Get
            Throw New Exception("The method or operation is not implemented.")
        End Get
    End Property

    Public ReadOnly Property CompletedSynchronously() As Boolean Implements
IAsyncResult.CompletedSynchronously
        Get
            Return True
        End Get
    End Property

    Public ReadOnly Property IsCompleted() As Boolean Implements IAsyncResult.IsCompleted
        Get
            Return True
        End Get
    End Property
    #End Region
End Class
End Namespace

```

See Also

[Designing Service Contracts](#)

[Synchronous and Asynchronous Operations](#)

Reliable Services

5/5/2018 • 2 minutes to read • [Edit Online](#)

Queues and reliable sessions are the Windows Communication Foundation (WCF) features that implement reliable messaging. This topic explains the reliable messaging features of WCF.

Reliable messaging is how a reliable messaging source (called the *source*) transfers messages reliably to a reliable messaging destination (called the *destination*).

Reliable messaging performs the following functions:

- Transfers assurances for messages sent from a source to a destination regardless of message transfer or transport failures.
- Separates the source and the destination from each other. This provides independent failure and recovery of the source and the destination, as well as reliable transfer and delivery of messages, even when the source or destination is unavailable.

Reliable messaging frequently comes at the cost of high latency. *Latency* is the time it takes for the message to reach the destination from the source. WCF, therefore, provides the following types of reliable messaging:

- [Reliable Sessions](#), which offers reliable transfer without the cost of high latency.
- [Queues in WCF](#), which offers both reliable transfers and separation between the source and the destination.

Reliable Sessions

Reliable sessions provide end-to-end reliable transfer of messages between a source and a destination using the WS-Reliable Messaging protocol, regardless of the number or type of intermediaries that separate the messaging (source and destination) endpoints. This includes any transport intermediaries that do not use SOAP (for example, HTTP proxies) or intermediaries that use SOAP (for example, SOAP-based routers or bridges) that are required for messages to flow between the endpoints. Reliable sessions use an in-memory transfer window to mask SOAP message-level failures and re-establish connections in the case of transport failures.

Reliable sessions provide low-latency reliable message transfers. They provide for SOAP messages over any proxies or intermediaries, equivalent to what TCP provides for packets over IP bridges. For more information about reliable sessions, see [Reliable Sessions](#).

Queues

Queues in WCF provide both reliable transfers of messages and separation between sources and destinations at the cost of high latency. WCF queued communication is built on top of Message Queuing (MSMQ).

MSMQ ships as an optional component with Windows. The MSMQ service runs as a Windows Service. It captures messages for transmission in a transmission queue on behalf of the source and delivers it to a target queue. The target queue accepts messages on behalf of the destination for later delivery whenever the destination requests messages. The MSMQ managers implement a reliable message-transfer protocol so that messages are not lost in transmission. The protocol can be native or a SOAP-based protocol called SOAP Reliable Messaging Protocol (SRMP).

The separation, coupled with reliable message transfers between queues, enables applications that are loosely coupled to communicate reliably. Unlike reliable sessions, the source and destination do not have to be running at the same time. This implicitly enables scenarios where queues are, in effect, used as a load-leveling mechanism when the source's rate of message production and the destination's rate of the message consumption do not

match. For more information about queues, see [Queues in WCF](#).

See Also

[Reliable Sessions Overview](#)

[Queuing in WCF](#)

Services and Transactions

5/4/2018 • 2 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) applications can initiate a transaction from within a client and coordinate the transaction within the service operation. Clients can initiate a transaction and invoke several service operations and ensure that the service operations are either committed or rolled back as a single unit.

You can enable the transaction behavior in the service contract by specifying a [ServiceBehaviorAttribute](#) and setting its [TransactionIsolationLevel](#) and [TransactionScopeRequired](#) properties for service operations that require client transactions. The [TransactionAutoComplete](#) parameter specifies whether the transaction in which the method executes is automatically completed if no unhandled exceptions are thrown. For more information about these attributes, see [ServiceModel Transaction Attributes](#).

The work that is performed in the service operations and managed by a resource manager, such as logging database updates, is part of the client's transaction.

The following sample demonstrates usage of the [ServiceBehaviorAttribute](#) and [OperationBehaviorAttribute](#) attributes to control service-side transaction behavior.

```
[ServiceBehavior(TransactionIsolationLevel = System.Transactions.IsolationLevel.Serializable)]
public class CalculatorService: ICalculatorLog
{
    [OperationBehavior(TransactionScopeRequired = true,
        TransactionAutoComplete = true)]
    public double Add(double n1, double n2)
    {
        recordToLog(String.Format("Added {0} to {1}", n1, n2));
        return n1 + n2;
    }

    [OperationBehavior(TransactionScopeRequired = true,
        TransactionAutoComplete = true)]
    public double Subtract(double n1, double n2)
    {
        recordToLog(String.Format("Subtracted {0} from {1}", n1, n2));
        return n1 - n2;
    }

    [OperationBehavior(TransactionScopeRequired = true,
        TransactionAutoComplete = true)]
    public double Multiply(double n1, double n2)
    {
        recordToLog(String.Format("Multiplied {0} by {1}", n1, n2));
        return n1 * n2;
    }

    [OperationBehavior(TransactionScopeRequired = true,
        TransactionAutoComplete = true)]
    public double Divide(double n1, double n2)
    {
        recordToLog(String.Format("Divided {0} by {1}", n1, n2));
        return n1 / n2;
    }
}
```

You can enable transactions and transaction flow by configuring the client and service bindings to use the WS-AtomicTransaction protocol, and setting the [transactionFlow](#) element to `true`, as shown in the following

sample configuration.

```
<client>
  <endpoint address="net.tcp://localhost/ServiceModelSamples/service"
    binding="netTcpBinding"
    bindingConfiguration="netTcpBindingWSAT"
    contract="Microsoft.ServiceModel.Samples.ICalculatorLog" />
</client>

<bindings>
  <netTcpBinding>
    <binding name="netTcpBindingWSAT"
      transactionFlow="true"
      transactionProtocol="WSAtomicTransactionOctober2004" />
    </netTcpBinding>
  </bindings>
```

Clients can begin a transaction by creating a [TransactionScope](#) and invoking service operations within the scope of the transaction.

```
using (TransactionScope ts = new TransactionScope(TransactionScopeOption.RequiresNew))
{
    //Do work here
    ts.Complete();
}
```

See Also

[Transactional Support in System.ServiceModel](#)

[Transaction Models](#)

[WS Transaction Flow](#)

Implementing Service Contracts

5/4/2018 • 2 minutes to read • [Edit Online](#)

A service is a class that exposes functionality available to clients at one or more endpoints. To create a service, write a class that implements a Windows Communication Foundation (WCF) contract. You can do this in one of two ways. You can define the contract separately as an interface and then create a class that implements that interface. Alternatively, you can create the class and contract directly by placing the [ServiceContractAttribute](#) attribute on the class itself and the [OperationContractAttribute](#) attribute on the methods available to the clients of the service.

Creating a service class

The following is an example of a service that implements an `IMath` contract that has been defined separately.

```
// Define the IMath contract.
[ServiceContract]
public interface IMath
{
    [OperationContract]
    double Add(double A, double B);

    [OperationContract]
    double Multiply (double A, double B);
}

// Implement the IMath contract in the MathService class.
public class MathService : IMath
{
    public double Add (double A, double B) { return A + B; }
    public double Multiply (double A, double B) { return A * B; }
}
```

Alternatively, a service can expose a contract directly. The following is an example of a service class that defines and implements a `MathService` contract.

```
// Define the MathService contract directly on the service class.
[ServiceContract]
class MathService
{
    [OperationContract]
    public double Add(double A, double B) { return A + B; }
    [OperationContract]
    private double Multiply (double A, double B) { return A * B; }
}
```

Note that the preceding services expose different contracts because the contract names are different. In the first case, the exposed contract is named "`IMath`" while in the second case the contract is named "`MathService`".

You can set a few things at the service and operation implementation levels, such as concurrency and instancing. For more information, see [Designing and Implementing Services](#).

After implementing a service contract, you must create one or more endpoints for the service. For more information, see [Endpoint Creation Overview](#). For more information about how to run a service, see [Hosting Services](#).

See Also

[Designing and Implementing Services](#)

[How to: Create a Service with a Contract Class](#)

[How to: Create a Service with a Contract Interface](#)

[Specifying Service Run-Time Behavior](#)

Specifying Service Run-Time Behavior

10/27/2018 • 6 minutes to read • [Edit Online](#)

Once you have designed a service contract ([Designing Service Contracts](#)) and implemented your service contract ([Implementing Service Contracts](#)) you can configure the operation behavior of the service runtime. This topic discusses system-provided service and operation behaviors and describes where to find more information to create new behaviors. While some behaviors are applied as attributes, many are applied using an application configuration file or programmatically. For more information about configuring your service application, see [Configuring Services](#).

Overview

The contract defines the inputs, outputs, data types, and capabilities of a service of that type. Implementing a service contract creates a class that, when configured with a binding at an address, fulfills the contract it implements. Contractual, binding, and address information are all known by the client; without them, the client cannot make use of the service.

However, operation specifics, such as threading issues or instance management, are opaque to clients. Once you have implemented your service contract, you can configure a large number of operation characteristics by using *behaviors*. Behaviors are objects that modify the Windows Communication Foundation (WCF) runtime by either setting a runtime property or by inserting a customization type into the runtime. For more information about modifying the runtime by creating user-defined behaviors, see [Extending ServiceHost and the Service Model Layer](#).

The [System.ServiceModel.ServiceBehaviorAttribute](#) and [System.ServiceModel.OperationBehaviorAttribute](#) attributes are the most widely useful behaviors and expose the most commonly requested operation features. Because they are attributes, you apply them to the service or operation implementation. Other behaviors, such as the [System.ServiceModel.Description.ServiceMetadataBehavior](#) or [System.ServiceModel.Description.ServiceDebugBehavior](#), are typically applied using an application configuration file, although you can use them programmatically.

This topic provides an overview of the [ServiceBehaviorAttribute](#) and [OperationBehaviorAttribute](#) attributes, describes the various scopes at which behaviors can operate, and provides a quick description of many of the system-provided behaviors at the various scopes that may be of interest to WCF developers.

ServiceBehaviorAttribute and OperationBehaviorAttribute

The most important behaviors are the [ServiceBehaviorAttribute](#) and [OperationBehaviorAttribute](#) attributes, which you can use to control:

- Instance lifetimes
- Concurrency and synchronization support
- Configuration behavior
- Transaction behavior
- Serialization behavior
- Metadata transformation
- Session lifetime

- Address filtering and header processing
- Impersonation
- To use these attributes, mark the service or operation implementation with the attribute appropriate to that scope and set the properties. For example, the following code example shows an operation implementation that uses the [OperationBehaviorAttribute.Impersonation](#) property to require that callers of this operation support impersonation.

```
using System;
using System.Collections.Generic;
using System.ServiceModel;
using System.Threading;

namespace Microsoft.WCF.Documentation
{
    [ServiceContract(
        Name="SampleHello",
        Namespace="http://microsoft.wcf.documentation"
    )]
    public interface IHello
    {
        [OperationContract]
        string Hello(string greeting);
    }

    public class HelloService : IHello
    {
        public HelloService()
        {
            Console.WriteLine("Service object created: " + this.GetHashCode().ToString());
        }

        ~HelloService()
        {
            Console.WriteLine("Service object destroyed: " + this.GetHashCode().ToString());
        }

        [OperationBehavior(Impersonation=ImpersonationOption.Required)]
        public string Hello(string greeting)
        {
            Console.WriteLine("Called by: " + Thread.CurrentPrincipal.Identity.Name);
            Console.WriteLine("IsAuthenticated: " + Thread.CurrentPrincipal.Identity.IsAuthenticated.ToString());
            Console.WriteLine("AuthenticationType: " +
                Thread.CurrentPrincipal.Identity.AuthenticationType.ToString());

            Console.WriteLine("Caller sent: " + greeting);
            Console.WriteLine("Sending back: Hi, " + Thread.CurrentPrincipal.Identity.Name);
            return "Hi, " + Thread.CurrentPrincipal.Identity.Name;
        }
    }
}
```

```
Imports System.ServiceModel
Imports System.Threading

Namespace Microsoft.WCF.Documentation
    <ServiceContract(Name="SampleHello", Namespace="http://microsoft.wcf.documentation")> _
    Public Interface IHello
        <OperationContract> _
        Function Hello(ByVal greeting As String) As String
    End Interface

    Public Class HelloService
        Implements IHello

        Public Sub New()
            Console.WriteLine("Service object created: " & Me.GetHashCode().ToString())
        End Sub

        Protected Overrides Sub Finalize()
            Console.WriteLine("Service object destroyed: " & Me.GetHashCode().ToString())
        End Sub

        <OperationBehavior(Impersonation:=ImpersonationOption.Required)> _
        Public Function Hello(ByVal greeting As String) As String Implements IHello.Hello
            Console.WriteLine("Called by: " & Thread.CurrentPrincipal.Identity.Name)
            Console.WriteLine("IsAuthenticated: " & Thread.CurrentPrincipal.Identity.IsAuthenticated.ToString())
            Console.WriteLine("AuthenticationType: " &
                Thread.CurrentPrincipal.Identity.AuthenticationType.ToString())

            Console.WriteLine("Caller sent: " & greeting)
            Console.WriteLine("Sending back: Hi, " & Thread.CurrentPrincipal.Identity.Name)
            Return "Hi, " & Thread.CurrentPrincipal.Identity.Name
        End Function
    End Class
End Namespace
```

Many of the properties require additional support from the binding. For example, an operation that requires a transaction from the client must be configured to use a binding that supports flowed transactions.

Well-Known Singleton Services

You can use the [ServiceBehaviorAttribute](#) and [OperationBehaviorAttribute](#) attributes to control certain lifetimes, both of the [InstanceContext](#) and of the service objects that implement the operations.

For example, the [ServiceBehaviorAttribute.InstanceContextMode](#) property controls how often the [InstanceContext](#) is released, and the [OperationBehaviorAttribute.ReleaseInstanceMode](#) and [ServiceBehaviorAttribute.ReleaseServiceInstanceOnTransactionComplete](#) properties control when the service object is released.

However, you can also create a service object yourself and create the service host using that object. To do so, you must also set the [ServiceBehaviorAttribute.InstanceContextMode](#) property to [Single](#) or an exception is thrown when the service host is opened.

Use the [ServiceHost.ServiceHost\(Object, Uri\[\]\)](#) constructor to create such a service. It provides an alternative to implementing a custom [System.ServiceModel.Dispatcher.InstanceContextInitializer](#) when you wish to provide a specific object instance for use by a singleton service. You can use this overload when your service implementation type is difficult to construct (for example, if it does not implement a default public constructor that has no parameters).

Note that when an object is provided to this constructor, some features related to the Windows Communication Foundation (WCF) instancing behavior work differently. For example, calling [InstanceContext.ReleaseServiceInstance](#) has no effect when a well-known object instance is provided. Similarly, any other instance release mechanism is ignored. The [ServiceHost](#) class always behaves as if the

[OperationBehaviorAttribute.ReleaseInstanceMode](#) property is set to [ReleaseInstanceMode.None](#) for all operations.

Other Service, Endpoint, Contract, and Operation Behaviors

Service behaviors, such as the [ServiceBehaviorAttribute](#) attribute, operate across an entire service. For example, if you set the [ServiceBehaviorAttribute.ConcurrencyMode](#) property to [ConcurrencyMode.Multiple](#) you must handle thread synchronization issues inside each operation in that service yourself. Endpoint behaviors operate across an endpoint; many of the system-provided endpoint behaviors are for client functionality. Contract behaviors operate at the contract level, and operation behaviors modify operation delivery.

Many of these behaviors are implemented on attributes, and you make use of them as you do the [ServiceBehaviorAttribute](#) and [OperationBehaviorAttribute](#) attributes—by applying them to the appropriate service class or operation implementation. Other behaviors, such as the [ServiceMetadataBehavior](#) or [ServiceDebugBehavior](#) objects, are typically applied using an application configuration file, although they can also be used programmatically.

For example, the publication of metadata is configured by using the [ServiceMetadataBehavior](#) object. The following application configuration file shows the most common usage.

```
<configuration>
  <system.serviceModel>
    <services>
      <service
        name="Microsoft.WCF.Documentation.SampleService"
        behaviorConfiguration="metadataSupport"
      >
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8080/SampleService" />
          </baseAddresses>
        </host>
        <endpoint
          address=""
          binding="wsHttpBinding"
          contract="Microsoft.WCF.Documentation.ISampleService"
        />
        <!-- Adds a WS-MetadataExchange endpoint at -->
        <!-- "http://localhost:8080/SampleService/mex" -->
        <endpoint
          address="mex"
          binding="mexHttpBinding"
          contract="IMetadataExchange"
        />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="metadataSupport">
          <!-- Enables the IMetadataExchange endpoint in services that -->
          <!-- use "metadataSupport" in their behaviorConfiguration attribute. -->
          <!-- In addition, the httpGetEnabled and httpGetUrl attributes publish -->
          <!-- Service metadata for retrieval by HTTP/GET at the address -->
          <!-- "http://localhost:8080/SampleService?wsdl" -->
          <serviceMetadata httpGetEnabled="true" httpGetUrl="" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

The following sections describe many of the most useful system-provided behaviors that you can use to modify

the runtime delivery of your service or client. See the reference topic to determine how to use each one.

Service Behaviors

The following behaviors operate on services.

- [AspNetCompatibilityRequirementsAttribute](#). Applied to a WCF service to indicate whether that service can be run in ASP.NET Compatibility Mode.
- [ServiceAuthorizationBehavior](#). Controls how the service authorizes client claims.
- [ServiceCredentials](#). Configures a service credential. Use this class to specify the credential for the service, such as an X.509 certificate.
- [ServiceDebugBehavior](#). Enables debugging and Help information features for a WCF service.
- [ServiceMetadataBehavior](#). Controls the publication of service metadata and associated information.
- [ServiceSecurityAuditBehavior](#). Specifies the audit behavior of security events.
- [ServiceThrottlingBehavior](#). Configures run-time throughput settings that enable you to tune service performance.

Endpoint Behaviors

The following behaviors operate on endpoints. Many of these behaviors are used in client applications.

- [CallbackBehaviorAttribute](#). Configures a callback service implementation in a duplex client application.
- [CallbackDebugBehavior](#). Enables service debugging for a WCF callback object.
- [ClientCredentials](#). Allows the user to configure client and service credentials as well as service credential authentication settings for use on the client.
- [ClientViaBehavior](#). Used by clients to specify the Uniform Resource Identifier (URI) for which the transport channel should be created.
- [MustUnderstandBehavior](#). Instructs WCF to disable the `MustUnderstand` processing.
- [SynchronousReceiveBehavior](#). Instructs the runtime to use a synchronous receive process for channels.
- [TransactedBatchingBehavior](#). Optimizes the receive operations for transports that support transactional receives.

Contract Behaviors

[DeliveryRequirementsAttribute](#). Specifies the feature requirements that bindings must provide to the service or client implementation.

Operation Behaviors

The following operation behaviors specify serialization and transaction controls for operations.

- [DataContractSerializerOperationBehavior](#). Represents the run-time behavior of the `System.Runtime.Serialization.DataContractSerializer`.
- [XmlSerializerOperationBehavior](#). Controls run-time behavior of the `XmlSerializer` and associates it with an operation.
- [TransactionFlowAttribute](#). Specifies the level in which a service operation accepts a transaction header.

See Also

[Configuring Services](#)

[How to: Control Service Instancing](#)

Configuring Services

5/5/2018 • 2 minutes to read • [Edit Online](#)

Once you have designed and implemented your service contract, you are ready to configure your service. This is where you define and customize how your service is exposed to clients, including specifying the address where it can be found, the transport and message encoding it uses to send and receive messages, and the type of security it requires.

Configuration as used here includes all the ways, imperatively in code or by using a configuration file, in which you can define and customize the various aspects of a service, such as specifying its endpoint addresses, the transports used, and its security schemes. In practice, writing configuration is a major part of programming WCF applications.

In This Section

[Simplified Configuration](#)

Starting with .NET Framework version 4, WCF comes with a new default configuration model that simplifies WCF configuration requirements. If you do not provide any WCF configuration for a particular service, the runtime automatically configures your service with default endpoints, bindings, and behaviors.

[Configuring Services Using Configuration Files](#)

A Windows Communication Foundation (WCF) service is configurable using the .NET Framework configuration technology. Most commonly, XML elements are added to the Web.config file for an Internet Information Services (IIS) site that hosts a WCF service. The elements allow you to change details, such as the endpoint addresses (the actual addresses used to communicate with the service) on a machine-by-machine basis.

[Bindings](#)

In addition, WCF includes several system-provided common configurations in the form of bindings that allow you to quickly select the most basic features for how a client and service communicate, such as the transports, security, and message encodings used.

[Endpoints](#)

All communication with a WCF service occurs through the *endpoints* of the service. Endpoints contain the contract, the configuration information that is specified in the bindings, and the addresses that indicate where to find the service or where to obtain information about the service.

[Securing Services](#)

Using WCF and existing security mechanisms, you can implement confidentiality, integrity, authentication, and authorization into any service. You can also audit for security successes and failures.

[Creating WS-I Basic Profile 1.1 Interoperable Services](#)

The requirements for deploying a service that is interoperable with services and clients on any other platform or operating system are outlined in the WS-I Basic Profile 1.1 specification.

Reference

[System.ServiceModel](#)

[System.ServiceModel.Channels](#)

[System.ServiceModel.Description](#)

Related Sections

[Basic Programming Lifecycle](#)

[Designing and Implementing Services](#)

[Hosting Services](#)

[Building Clients](#)

[Introduction to Extensibility](#)

[Administration and Diagnostics](#)

See Also

[Basic WCF Programming](#)

[Conceptual Overview](#)

[WCF Feature Details](#)

Simplified Configuration

9/18/2018 • 4 minutes to read • [Edit Online](#)

Configuring Windows Communication Foundation (WCF) services can be a complex task. There are many different options and it is not always easy to determine what settings are required. While configuration files increase the flexibility of WCF services, they also are the source for many hard to find problems. .NET Framework 4.6.1 addresses these problems and provides a way to reduce the size and complexity of service configuration.

Simplified Configuration

In WCF service configuration files, the `<system.serviceModel>` section contains a `<service>` element for each service hosted. The `<service>` element contains a collection of `<endpoint>` elements that specify the endpoints exposed for each service and optionally a set of service behaviors. The `<endpoint>` elements specify the address, binding, and contract exposed by the endpoint, and optionally binding configuration and endpoint behaviors. The `<system.serviceModel>` section also contains a `<behaviors>` element that allows you to specify service or endpoint behaviors. The following example shows the `<system.serviceModel>` section of a configuration file.

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior name="MyServiceBehavior">
        <serviceMetadata httpGetEnabled="true">
        </serviceMetadata>
        <serviceDebug includeExceptionDetailInFaults="false">
        </serviceDebug>
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <bindings>
    <basicHttpBinding>
      <binding name="MyBindingConfig"
        maxBufferSize="100"
        maxReceiveBufferSize="100" />
    </basicHttpBinding>
  </bindings>
  <services>
    <service behaviorConfiguration="MyServiceBehavior"
      name="MyService">
      <endpoint address=""
        binding="basicHttpBinding"
        contract="ICalculator"
        bindingConfiguration="MyBindingConfig" />
      <endpoint address="mex"
        binding="mexHttpBinding"
        contract="IMetadataExchange"/>
    </service>
  </services>
</system.serviceModel>
```

.NET Framework 4.6.1 makes configuring a WCF service easier by removing the requirement for the `<service>` element. If you do not add a `<service>` section or add any endpoints in a `<service>` section and your service does not programmatically define any endpoints, then a set of default endpoints are automatically added to your service, one for each service base address and for each contract implemented by your service. In each of these endpoints, the endpoint address corresponds to the base address, the binding is determined by the base address scheme and the contract is the one implemented by your service. If you do not need to specify any endpoints or service behaviors or make any binding setting changes, you do not need to specify a service configuration file at all. If a service implements two contracts and the host enables both HTTP and TCP transports the service host

creates four default endpoints, one for each contract using each transport. To create default endpoints the service host must know what bindings to use. These settings are specified in a `< protocolMappings >` section within the `< system.serviceModel >` section. The `< protocolMappings >` section contains a list of transport protocol schemes mapped to binding types. The service host uses the base addresses passed to it to determine which binding to use. The following example uses the `< protocolMappings >` element.

WARNING

Changing default configuration elements, such as bindings or behaviors, might affect services defined in lower levels of the configuration hierarchy, since they might be using these default bindings and behaviors. Thus, whoever changes default bindings and behaviors needs to be aware that these changes might affect other services in the hierarchy.

NOTE

Services hosted under Internet Information Services (IIS) or Windows Process Activation Service (WAS) use the virtual directory as their base address.

```
<protocolMapping>
  <add scheme="http"    binding="basicHttpBinding" bindingConfiguration="MyBindingConfiguration"/>
  <add scheme="net.tcp" binding="netTcpBinding"/>
  <add scheme="net.pipe" binding="netNamedPipeBinding"/>
  <add scheme="net.msmq" binding="netMSMQBinding"/>
</protocolMapping>
```

In the previous example, an endpoint with a base address that starts with the "http" scheme uses the [BasicHttpBinding](#). An endpoint with a base address that starts with the "net.tcp" scheme uses the [NetTcpBinding](#). You can override settings in a local App.config or Web.config file.

Each element within the `< protocolMappings >` section must specify a scheme and a binding. Optionally it can specify a `bindingConfiguration` attribute that specifies a binding configuration within the `< bindings >` section of the configuration file. If no `bindingConfiguration` is specified, the anonymous binding configuration of the appropriate binding type is used.

Service behaviors are configured for the default endpoints by using anonymous `< behavior >` sections within `< serviceBehaviors >` sections. Any unnamed `< behavior >` elements within `< serviceBehaviors >` are used to configure service behaviors. For example, the following configuration file enables service metadata publishing for all services within the host.

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior>
        <serviceMetadata httpGetEnabled="True"/>
      </behavior>
    </serviceBehaviors>
  </behaviors>    <!-- No <service> tag is necessary. Default endpoints are added to the service -->
  <!-- The service behavior with name="" is picked up by the service -->
</system.serviceModel>
```

Endpoint behaviors are configured by using anonymous `< behavior >` sections within `< serviceBehaviors >` sections.

The following example is a configuration file equivalent to the one at the beginning of this topic that uses the simplified configuration model.

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior>
        <serviceMetadata httpGetEnabled="true" />
        <serviceDebug includeExceptionDetailInFaults="false" />
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <bindings>
    <basicHttpBinding>
      <binding maxBufferSize="100"
        maxReceiveBufferSize="100" />
    </basicHttpBinding>
  </bindings>
  <!-- No <service> tag is necessary. Default endpoints will be added to the service -->
  <!-- The service behavior with name="" will be picked up by the service -->
  <protocolMapping>
    <add scheme="http" binding="basicHttpBinding" />
  </protocolMapping>
</system.serviceModel>
```

IMPORTANT

This feature relates only to WCF service configuration, not client configuration. Most times, WCF client configuration will be generated by a tool such as svcutil.exe or adding a service reference from Visual Studio. If you are manually configuring a WCF client you will need to add a <client> element to the configuration and specify any endpoints you wish to call.

See Also

[Configuring Services Using Configuration Files](#)

[Configuring Bindings for Services](#)

[Configuring System-Provided Bindings](#)

[Configuring Services](#)

[Configuring Windows Communication Foundation Applications](#)

[Configuring WCF Services in Code](#)

Configuring Services Using Configuration Files

10/27/2018 • 8 minutes to read • [Edit Online](#)

Configuring a Windows Communication Foundation (WCF) service with a configuration file gives you the flexibility of providing endpoint and service behavior data at the point of deployment instead of at design time. This topic outlines the primary techniques available.

A WCF service is configurable using the .NET Framework configuration technology. Most commonly, XML elements are added to the Web.config file for an Internet Information Services (IIS) site that hosts a WCF service. The elements allow you to change details such as the endpoint addresses (the actual addresses used to communicate with the service) on a machine-by-machine basis. In addition, WCF includes several system-provided elements that allow you to quickly select the most basic features for a service. Starting with .NET Framework version 4, WCF comes with a new default configuration model that simplifies WCF configuration requirements. If you do not provide any WCF configuration for a particular service, the runtime automatically configures your service with some standard endpoints and default binding/behavior. In practice, writing configuration is a major part of programming WCF applications.

For more information, see [Configuring Bindings for Services](#). For a list of the most commonly used elements, see [System-Provided Bindings](#). For more information about default endpoints, bindings, and behaviors, see [Simplified Configuration](#) and [Simplified Configuration for WCF Services](#).

IMPORTANT

When deploying side by side scenarios where two different versions of a service are deployed, it is necessary to specify partial names of assemblies referenced in configuration files. This is because the configuration file is shared across all versions of a service and they could be running under different versions of the .NET Framework.

System.Configuration: Web.config and App.config

WCF uses the System.Configuration configuration system of the .NET Framework.

When configuring a service in Visual Studio, use either a Web.config file or an App.config file to specify the settings. The choice of the configuration file name is determined by the hosting environment you choose for the service. If you are using IIS to host your service, use a Web.config file. If you are using any other hosting environment, use an App.config file.

In Visual Studio, the file named App.config is used to create the final configuration file. The final name actually used for the configuration depends on the assembly name. For example, an assembly named "Cohowinery.exe" has a final configuration file name of "Cohowinery.exe.config". However, you only need to modify the App.config file. Changes made to that file are automatically made to the final application configuration file at compile time.

In using an App.config, file the configuration system merges the App.config file with content of the Machine.config file when the application starts and the configuration is applied. This mechanism allows machine-wide settings to be defined in the Machine.config file. The App.config file can be used to override the settings of the Machine.config file; you can also lock in the settings in Machine.config file so that they get used. In the Web.config case, the configuration system merges the Web.config files in all directories leading up to the application directory into the configuration that gets applied. For more information about configuration and the setting priorities, see topics in the [System.Configuration](#) namespace.

Major Sections of the Configuration File

The main sections in the configuration file include the following elements.

```
<system.ServiceModel>

  <services>
  <!-- Define the service endpoints. This section is optional in the new
  default configuration model in .NET Framework 4. -->
    <service>
      <endpoint/>
    </service>
  </services>

  <bindings>
  <!-- Specify one or more of the system-provided binding elements,
  for example, <basicHttpBinding> -->
  <!-- Alternatively, <customBinding> elements. -->
    <binding>
      <!-- For example, a <BasicHttpBinding> element. -->
    </binding>
  </bindings>

  <behaviors>
  <!-- One or more of the system-provided or custom behavior elements. -->
    <behavior>
      <!-- For example, a <throttling> element. -->
    </behavior>
  </behaviors>

</system.ServiceModel>
```

NOTE

The bindings and behaviors sections are optional and are only included if required.

The <services> Element

The `services` element contains the specifications for all services the application hosts. Starting with the simplified configuration model in .NET Framework 4, this section is optional.

<services>

The <service> Element

Each service has these attributes:

- `name`. Specifies the type that provides an implementation of a service contract. This is a fully qualified name which consists of the namespace, a period, and then the type name. For example `"MyNameSpace.myServiceType"`.
- `behaviorConfiguration`. Specifies the name of one of the `behavior` elements found in the `behaviors` element. The specified behavior governs actions such as whether the service allows impersonation. If its value is the empty name or no `behaviorConfiguration` is provided then the default set of service behaviors is added to the service.

- <service>

The <endpoint> Element

Each endpoint requires an address, a binding, and a contract, which are represented by the following attributes:

- `address`. Specifies the service's Uniform Resource Identifier (URI), which can be an absolute address or one that is given relative to the base address of the service. If set to an empty string, it indicates that the endpoint is available at the base address that is specified when creating the [ServiceHost](#) for the service.

- `binding`. Typically specifies a system-provided binding like [WSHttpBinding](#), but can also specify a user-defined binding. The binding specified determines the type of transport, security and encoding used, and whether reliable sessions, transactions, or streaming is supported or enabled.
- `bindingConfiguration`. If the default values of a binding must be modified, this can be done by configuring the appropriate `binding` element in the `bindings` element. This attribute should be given the same value as the `name` attribute of the `binding` element that is used to change the defaults. If no name is given, or no `bindingConfiguration` is specified in the binding, then the default binding of the binding type is used in the endpoint.
- `contract`. Specifies the interface that defines the contract. This is the interface implemented in the common language runtime (CLR) type specified by the `name` attribute of the `service` element.
- [<endpoint> element reference](#)

The <bindings> Element

The `bindings` element contains the specifications for all bindings that can be used by any endpoint defined in any service.

[<bindings>](#)

The <binding> Element

The `binding` elements contained in the `bindings` element can be either one of the system-provided bindings (see [System-Provided Bindings](#)) or a custom binding (see [Custom Bindings](#)). The `binding` element has a `name` attribute that correlates the binding with the endpoint specified in the `bindingConfiguration` attribute of the `endpoint` element. If no name is specified then that binding corresponds to the default of that binding type.

For more information about configuring services and clients, see [Configuring Windows Communication Foundation Applications](#).

[<binding>](#)

The <behaviors> Element

This is a container element for the `behavior` elements that define the behaviors for a service.

[<behaviors>](#)

The <behavior> Element

Each `behavior` element is identified by a `name` attribute and provides either a system-provided behavior, such as `<throttling>`, or a custom behavior. If no name is given then that behavior element corresponds to the default service or endpoint behavior.

[<behavior>](#)

How to Use Binding and Behavior Configurations

WCF makes it easy to share configurations between endpoints using a reference system in configuration. Rather than directly assigning configuration values to an endpoint, binding-related configuration values are grouped in `bindingConfiguration` elements in the `<binding>` section. A binding configuration is a named group of settings on a binding. Endpoints can then reference the `bindingConfiguration` by name.

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="myBindingConfiguration1" closeTimeout="00:01:00" />
        <binding name="myBindingConfiguration2" closeTimeout="00:02:00" />
        <binding closeTimeout="00:03:00" /> <!-- Default binding for basicHttpBinding -->
      </basicHttpBinding>
    </bindings>
    <services>
      <service name="MyNamespace.myServiceType">
        <endpoint
          address="myAddress" binding="basicHttpBinding"
          bindingConfiguration="myBindingConfiguration1"
          contract="MyContract" />
        <endpoint
          address="myAddress2" binding="basicHttpBinding"
          bindingConfiguration="myBindingConfiguration2"
          contract="MyContract" />
        <endpoint
          address="myAddress3" binding="basicHttpBinding"
          contract="MyContract" />
      </service>
    </services>
  </system.serviceModel>
</configuration>

```

The `name` of the `bindingConfiguration` is set in the `<binding>` element. The `name` must be a unique string within the scope of the binding type—in this case the `<basicHttpBinding>`, or an empty value to refer to the default binding. The endpoint links to the configuration by setting the `bindingConfiguration` attribute to this string.

A `behaviorConfiguration` is implemented the same way, as illustrated in the following sample.

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <endpointBehaviors>
        <behavior name="myBehavior">
          <callbackDebug includeExceptionDetailInFaults="true" />
        </behavior>
      </endpointBehaviors>
      <serviceBehaviors>
        <behavior>
          <serviceMetadata httpGetEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service name="NewServiceType">
        <endpoint
          address="myAddress3" behaviorConfiguration="myBehavior"
          binding="basicHttpBinding"
          contract="MyContract" />
      </service>
    </services>
  </system.serviceModel>
</configuration>

```

Note that the default set of service behaviors are added to the service. This system allows endpoints to share common configurations without redefining the settings. If machine-wide scope is required, create the binding or

behavior configuration in Machine.config. The configuration settings are available in all App.config files. The [Configuration Editor Tool \(SvcConfigEditor.exe\)](#) makes it easy to create configurations.

Behavior Merge

The behavior merge feature makes it easier to manage behaviors when you want a set of common behaviors to be used consistently. This feature allows you to specify behaviors at different levels of the configuration hierarchy and have services inherit behaviors from multiple levels of the configuration hierarchy. To illustrate how this works assume you have the following virtual directory layout in IIS:

```
~\Web.config~\Service.svc~\Child\Web.config~\Child\Service.svc
```

And your `~\Web.config` file has the following contents:

```
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <serviceDebug includeExceptionDetailInFaults="True" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

And you have a child Web.config located at `~\Child\Web.config` with the following contents:

```
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <serviceMetadata httpGetEnabled="True" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

The service located at `~\Child\Service.svc` will behave as though it has both the `serviceDebug` and `serviceMetadata` behaviors. The service located at `~\Service.svc` will only have the `serviceDebug` behavior. What happens is that the two behavior collections with the same name (in this case the empty string) are merged.

You can also clear behavior collections by using the `<clear>` tag and removed individual behaviors from the collection by using the `<remove>` tag. For example, the following two configuration results in the child service having only the `serviceMetadata` behavior:


```

<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <remove name="serviceDebug"/>
          <serviceMetadata httpGetEnabled="True" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>

```

```

<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <clear/>
          <serviceMetadata httpGetEnabled="True" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>

```

Behavior merge is done for nameless behavior collections as shown above and named behavior collections as well.

Behavior merge works in the IIS hosting environment, in which Web.config files merge hierarchically with the root Web.config file and machine.config. But it also works in the application environment, where machine.config can merge with the App.config file.

Behavior merge applies to both endpoint behaviors and service behaviors in configuration.

If a child behavior collection contains a behavior that's already present in the parent behavior collection, the child behavior overrides the parent. So if a parent behavior collection had `<serviceMetadata httpGetEnabled="False" />` and a child behavior collection had `<serviceMetadata httpGetEnabled="True" />`, the child behavior would override the parent behavior in the behavior collection and httpGetEnabled would be "true".

See Also

[Simplified Configuration](#)

[Configuring Windows Communication Foundation Applications](#)

[<service>](#)

[<binding>](#)

Configuring WCF Services in Code

9/26/2018 • 3 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) allows developers to configure services using configuration files or code. Configuration files are useful when a service needs to be configured after being deployed. When using configuration files, an IT professional only needs to update the configuration file, no recompilation is required. Configuration files, however, can be complex and difficult to maintain. There is no support for debugging configuration files and configuration elements are referenced by names which makes authoring configuration files error-prone and difficult. WCF also allows you to configure services in code. In earlier versions of WCF (4.0 and earlier) configuring services in code was easy in self-hosted scenarios, the [ServiceHost](#) class allowed you to configure endpoints and behaviors prior to calling `ServiceHost.Open`. In web hosted scenarios, however, you don't have direct access to the [ServiceHost](#) class. To configure a web hosted service you were required to create a `System.ServiceModel.ServiceHostFactory` that created the [ServiceHostFactory](#) and performed any needed configuration. Starting with .NET 4.5, WCF provides an easier way to configure both self-hosted and web hosted services in code.

The Configure method

Simply define a public static method called `Configure` with the following signature in your service implementation class:

```
public static void Configure(ServiceConfiguration config)
```

The `Configure` method takes a [ServiceConfiguration](#) instance that enables the developer to add endpoints and behaviors. This method is called by WCF before the service host is opened. When defined, any service configuration settings specified in an `app.config` or `web.config` file will be ignored.

The following code snippet illustrates how to define the `Configure` method and add a service endpoint, an endpoint behavior and service behaviors:

```

public class Service1 : IService1
{
    public static void Configure(ServiceConfiguration config)
    {
        ServiceEndpoint se = new ServiceEndpoint(new ContractDescription("IService1"), new
BasicHttpBinding(), new EndpointAddress("basic"));
        se.Behaviors.Add(new MyEndpointBehavior());
        config.AddServiceEndpoint(se);

        config.Description.Behaviors.Add(new ServiceMetadataBehavior { HttpGetEnabled = true });
        config.Description.Behaviors.Add(new ServiceDebugBehavior { IncludeExceptionDetailInFaults = true
});
    }

    public string GetData(int value)
    {
        return string.Format("You entered: {0}", value);
    }

    public CompositeType GetDataUsingDataContract(CompositeType composite)
    {
        if (composite == null)
        {
            throw new ArgumentNullException("composite");
        }
        if (composite.BoolValue)
        {
            composite.StringValue += "Suffix";
        }
        return composite;
    }
}

```

To enable a protocol such as https for a service, you can either explicitly add an endpoint that uses the protocol or you can automatically add endpoints by calling `ServiceConfiguration.EnableProtocol(Binding)` which adds an endpoint for each base address compatible with the protocol and each service contract defined. The following code illustrates how to use the `ServiceConfiguration.EnableProtocol` method:

```

public class Service1 : IService1
{
    public string GetData(int value);
    public static void Configure(ServiceConfiguration config)
    {
        // Enable "Add Service Reference" support
        config.Description.Behaviors.Add( new ServiceMetadataBehavior { HttpGetEnabled = true });
        // set up support for http, https, net.tcp, net.pipe
        config.EnableProtocol(new BasicHttpBinding());
        config.EnableProtocol(new BasicHttpBinding());
        config.EnableProtocol(new NetTcpBinding());
        config.EnableProtocol(new NetNamedPipeBinding());
        // add an extra BasicHttpBinding endpoint at http://basic
        config.AddServiceEndpoint(typeof(IService1), new BasicHttpBinding(),"basic");
    }
}

```

The settings in the < `protocolMappings` > section are only used if no application endpoints are added to the [ServiceConfiguration](#) programmatically. You can optionally load the service configuration from the default application configuration file by calling [LoadFromConfiguration](#) and then change the settings. The [LoadFromConfiguration\(\)](#) class also allows you to load configuration from a centralized configuration. The following code illustrates how to implement this:

```
public class Service1 : IService1
{
    public void DoWork();
    public static void Configure(ServiceConfiguration config)
    {
        config.LoadFromConfiguration(ConfigurationManager.OpenMappedExeConfiguration(new
ExeConfigurationFileMap { ExeConfigFilename = @"c:\sharedConfig\MyConfig.config" },
ConfigurationUserLevel.None));
    }
}
```

IMPORTANT

Note that [LoadFromConfiguration](#) ignores < host > settings within the < service > tag of < system.serviceModel >. Conceptually, < host > is about host configuration, not service configuration, and it gets loaded before the Configure method executes.

See Also

[Configuring Services Using Configuration Files](#)

[Configuring Client Behaviors](#)

[Simplified Configuration](#)

[Configuration](#)

[Configuration-Based Activation in IIS and WAS](#)

[Configuration and Metadata Support](#)

[Configuration](#)

[How to: Specify a Service Binding in Configuration](#)

[How to: Create a Service Endpoint in Configuration](#)

[How to: Publish Metadata for a Service Using a Configuration File](#)

[How to: Specify a Client Binding in Configuration](#)

Windows Communication Foundation Bindings

5/5/2018 • 2 minutes to read • [Edit Online](#)

Bindings specify how a Windows Communication Foundation (WCF) service endpoint communicates with other endpoints. At its most basic, a binding must specify the transport (for example, HTTP or TCP) to use. You can also set other characteristics, such as security and transaction support, through bindings.

In This Section

[WCF Bindings Overview](#)

Overview of what WCF bindings do, what bindings the system provides, and how you can define or modify them.

[System-Provided Bindings](#)

A list of bindings included with WCF. These bindings cover the majority of security and message pattern requirements.

[Using Bindings to Configure Services and Clients](#)

A WCF binding contains important information that clients must use to connect to service endpoints.

[Configuring Bindings for Services](#)

Configuration enables administrators and installers to customize the bindings for service endpoints.

Reference

[System.ServiceModel.Channels](#)

Related Sections

[Endpoints: Addresses, Bindings, and Contracts](#)

[Bindings](#)

See Also

[Custom Bindings](#)

Windows Communication Foundation Bindings Overview

8/7/2018 • 3 minutes to read • [Edit Online](#)

Bindings are objects that are used to specify the communication details that are required to connect to the endpoint of a Windows Communication Foundation (WCF) service. Each endpoint in a WCF service requires a binding to be well-specified. This topic outlines the types of communication details that the bindings define, the elements of a binding, what bindings are included in WCF, and how a binding can be specified for an endpoint.

What a Binding Defines

The information in a binding can be very basic, or very complex. The most basic binding specifies only the transport protocol (such as HTTP) that must be used to connect to the endpoint. More generally, the information a binding contains about how to connect to an endpoint falls into one of the following categories.

Protocols

Determines the security mechanism being used: either reliable messaging capability or transaction context flow settings.

Encoding

Determines the message encoding (for example, text or binary).

Transport

Determines the underlying transport protocol to use (for example, TCP or HTTP).

The Elements of a Binding

A binding basically consists of an ordered stack of binding elements, each of which specifies part of the communication information required to connect to a service endpoint. The two lowest layers in the stack are both required. At the base of the stack is the transport binding element and just above this is the element that contains the message encoding specifications. The optional binding elements that specify the other communication protocols are layered above these two required elements. For more information about these binding elements and their correct ordering, see [Custom Bindings](#).

System-Provided Bindings

The information in a binding can be complex, and some settings may not be compatible with others. For this reason, WCF includes a set of system-provided bindings. These bindings are designed to cover most application requirements. The following classes represent some examples of system-provided bindings:

- [BasicHttpBinding](#): An HTTP protocol binding suitable for connecting to Web services that conforms to the WS-I Basic Profile specification (for example, ASP.NET Web services-based services).
- [WSHttpBinding](#): An interoperable binding suitable for connecting to endpoints that conform to the WS-* protocols.
- [NetNamedPipeBinding](#): Uses the .NET Framework to connect to other WCF endpoints on the same machine.
- [NetMsmqBinding](#): Uses the .NET Framework to create queued message connections with other WCF endpoints.

- [NetTcpBinding](#): This binding offers higher performance than HTTP bindings and is ideal for use in a local network.

For a complete list, with descriptions, of all the WCF-provided bindings, see [System-Provided Bindings](#).

Using Your Own Bindings

If none of the system-provided bindings included has the right combination of features that a service application requires, you can create your own binding. There are two ways to do this. You can either create a new binding from pre-existing binding elements using a [CustomBinding](#) object or you can create a completely user-defined binding by deriving from the [Binding](#) binding. For more information about creating your own binding using these two approaches, see [Custom Bindings](#) and [Creating User-Defined Bindings](#).

Using Bindings

Using bindings entails two basic steps:

1. Select or define a binding. The easiest method is to choose one of the system-provided bindings included with WCF and use it with its default settings. You can also choose a system-provided binding and reset its property values to suit your requirements. Alternatively, you can create a custom binding or a user-defined binding to have higher degrees of control and customization.
2. Create an endpoint that uses the binding selected or defined.

Code and Configuration

You can define bindings in two ways: through code or through configuration. These two approaches do not depend on whether you are using a system-provided binding or a custom binding. In general, using code gives you complete control over the definition of a binding at design time. Using configuration, on the other hand, allows a system administrator or the user of a WCF service or client to change the parameters of a binding without having to recompile the service application. This flexibility is often desirable because there is no way to predict specific machine requirements on which a WCF application is to be deployed. Keeping the binding (and the addressing) information out of the code allows them to change without requiring recompilation or redeployment of the application. Note that bindings defined in code are created after bindings specified in configuration, allowing the code-defined bindings to overwrite any configuration-defined bindings.

See Also

[Using Bindings to Configure Services and Clients](#)

System-provided bindings

6/6/2018 • 5 minutes to read • [Edit Online](#)

Bindings specify the communication mechanism to use when talking to an endpoint and indicate how to connect to an endpoint. A binding contains the following elements:

- The protocol stack determines the security, reliability, and context flow settings to use for messages that are sent to the endpoint.
- The transport determines the underlying transport protocol to use when sending messages to the endpoint, for example, TCP or HTTP.
- The encoding determines the wire encoding to use for messages that are sent to the endpoint. For example, text/XML, binary, or Message Transmission Optimization Mechanism (MTOM).

This article presents all of the system-provided Windows Communication Foundation (WCF) bindings. If none of these bindings meets the exact criteria for your application, you can create a custom binding. For more information about creating custom bindings, see [Custom Bindings](#).

A secure and interoperable binding that supports the WS-Federation protocol enables organizations that are in a federation to efficiently authenticate and authorize users.

IMPORTANT

Always select a binding that includes security. By default, all bindings except the `<basicHttpBinding>` element have security enabled. If you do not select a secure binding or disable security, be sure to protect your data in some other manner, such as storing in a secured data center or on an isolated network.

IMPORTANT

Never use duplex contracts with bindings that do not support security or that have security disabled unless you secure the data by some other means.

The following bindings ship with WCF:

BINDING	CONFIGURATION ELEMENT	DESCRIPTION
BasicHttpBinding	<code><basicHttpBinding></code>	A binding that is suitable for communicating with WS-Basic Profile-conformant Web services, for example, ASP.NET Web services (ASMX)-based services. This binding uses HTTP as the transport and text/XML as the default message encoding.
WSHttpBinding	<code><wsHttpBinding></code>	A secure and interoperable binding that is suitable for non-duplex service contracts.

BINDING	CONFIGURATION ELEMENT	DESCRIPTION
WSDualHttpBinding	<wsDualHttpBinding>	A secure and interoperable binding that is suitable for duplex service contracts or communication through SOAP intermediaries.
WSFederationHttpBinding	<wsFederationHttpBinding>	A secure and interoperable binding that supports the WS-Federation protocol, which enables organizations that are in a federation to efficiently authenticate and authorize users.
NetHttpBinding	<netHttpBinding>	A binding designed for consuming HTTP or WebSocket services that uses binary encoding by default.
NetHttpsBinding	<netHttpsBinding>	A secure binding designed for consuming HTTP or WebSocket services that uses binary encoding by default.
NetTcpBinding	<netTcpBinding>	A secure and optimized binding suitable for cross-machine communication between WCF applications.
NetNamedPipeBinding	<netNamedPipeBinding>	A secure, reliable, optimized binding that is suitable for on-machine communication between WCF applications.
NetMsmqBinding	<netMsmqBinding>	A queued binding that is suitable for cross-machine communication between WCF applications.
NetPeerTcpBinding	<netPeerTcpBinding>	A binding that enables secure, multiple machine communication.
MsmqIntegrationBinding	<msmqIntegrationBinding>	A binding that is suitable for cross-machine communication between a WCF application and existing Message Queuing applications.
BasicHttpContextBinding	<basicHttpContextBinding>	A binding suitable for communicating with WS-Basic Profile conformant Web services that enables HTTP cookies to be used to exchange context.
NetTcpContextBinding	<netTcpContextBinding>	A secure and optimized binding suitable for cross-machine communication between WCF applications that enables SOAP headers to be used to exchange context.

BINDING	CONFIGURATION ELEMENT	DESCRIPTION
WebHttpBinding	<webHttpBinding>	A binding used to configure endpoints for WCF Web services that are exposed through HTTP requests instead of SOAP messages.
WSHttpContextBinding	<wsHttpContextBinding>	A secure and interoperable binding suitable for non-duplex service contracts that enables SOAP headers to be used to exchange context.
UdpBinding	<udpBinding>	A binding to use when sending a burst of simple messages to a large number of clients simultaneously.

The following table shows the features of each of the system-provided bindings. The bindings are found in the table columns; the features are listed in the rows and described in a second table. The following table provides a key for the binding abbreviations used. To select a binding, determine which column satisfies all of the row features you need.

BINDING	INTEROPERABILITY	SECURITY (DEFAULT)	SESSION (DEFAULT)	TRANSACTIONS	DUPLEX	ENCODING (DEFAULT)	STREAMING (DEFAULT)
BasicHttpBinding	Basic Profile 1.1	(None), Transport, Message, Mixed	(None)	(None)	n/a	Text, (MTOM)	Yes (buffered)
WSHttpBinding	WS	Transport, (Message), Mixed	(None), Reliable Session, Security Session	(None), Yes	n/a	(Text), MTOM	No
WSDualHttpBinding	WS	(Message), None	(Reliable Session), Security Session	(None), Yes	Yes	(Text), MTOM	No
WSFederationHttpBinding	WS-Federation	(Message), Mixed, None	(None), Reliable Session, Security Session	(None), Yes	No	(Text), MTOM	No
NetHttpBinding	.NET	(None), Transport, Message, TransportWithMessageCredential, TransportCredentialOnly	See note below	None	See note below	(Binary), Text, MTOM	Yes (buffered)

BINDING	INTEROPERABILITY	SECURITY (DEFAULT)	SESSION (DEFAULT)	TRANSACTIONS	DUPLEX	ENCODING (DEFAULT)	STREAMING (DEFAULT)
NetHttpsBinding	.NET	(Transport), TransportWithMessageCredential	See note below	None	See note below	(Binary), Text, MTOM	Yes (buffered)
NetTcpBinding	.NET	(Transport), Message, None, Mixed	(Transport), Reliable Session, Security Session	(None), Yes	Yes	Binary	Yes (buffered)
NetNamedPipeBinding	.NET	(Transport), None	None, (Transport)	(None), Yes	Yes	Binary	Yes (buffered)
NetMsmqBinding	.NET	Message, (Transport), None	(None), Transport	None, (Yes)	No	Binary	No
NetPeerTcpBinding	Peer	(Transport)	(None)	(None)	Yes		No
MsmqIntegrationBinding	MSMQ	(Transport)	(None)	None, (Yes)	n/a	n/a	No
BasicHttpContextBinding	Basic Profile 1.1	(None), Transport, Message, Mixed	(None)	(None)	n/a	Text, (MTOM)	Yes (buffered)
NetTcpContextBinding	.NET	(Transport), Message, None, Mixed	(Transport), Reliable Session, Security Session	(None), Yes	Yes	Binary	Yes (buffered)
WSHttpContextBinding	WS	Transport, (Message), Mixed	(None), Reliable Session, Security Session	(None), Yes	n/a	Text, (MTOM)	No

BINDING	INTEROPERA BILITY	SECURITY (DEFAULT)	SESSION (DEFAULT)	TRANSACTION S	DUPLEX	ENCODING (DEFAULT)	STREAMING (DEFAULT)
UdpBinding Note: Interoperability can be achieved by implementing the standard SOAP-over-UDP spec which this binding implements.	.NET	(None)	(None)	(None)	n/a	(Text)	No

IMPORTANT

[NetHttpBinding](#) is a binding designed for consuming HTTP or WebSocket services and uses binary encoding by default. [NetHttpBinding](#) detects whether it's used with a request-reply contract or duplex contract and changes its behavior to match; it uses HTTP for request-reply and WebSockets for duplex. This behavior can be overridden using the [WebSocketTransportUsage](#) binding setting: WhenDuplex - This is the default value and behaves as described above. Never - This prevents WebSockets from being used. Attempting to use a duplex contract with this setting results in an exception. Always - This forces WebSockets to be used even for request-reply contracts. [NetHttpBinding](#) supports reliable sessions in both HTTP mode and WebSocket mode. In WebSocket mode sessions are provided by the transport.

The following table explains the features listed in the previous table.

FEATURE	DESCRIPTION
Interoperability Type	Names the protocol or technology with which the binding ensures interoperation.
Security	Specifies how the channel is secured: <ul style="list-style-type: none"> - None: The SOAP message isn't secured and the client isn't authenticated. - Transport: Security requirements are satisfied at the transport layer. - Message: Security requirements are satisfied at the message layer. - Mixed: Claims are carried in the message; integrity and confidentiality requirements are satisfied by the transport layer.
Session	Specifies whether this binding supports session contracts.
Transactions	Specifies whether transactions are enabled.
Duplex	Specifies whether duplex contracts are supported. Note that this feature requires support for Sessions in the binding.

FEATURE	DESCRIPTION
Encoding	<p>Specifies the wire format of the message. Allowable values include:</p> <ul style="list-style-type: none"> - Text: for example UTF-8. - Binary - Message Transmission Optimization Mechanism (MTOM): A method for efficiently encoding binary XML elements within the context of a SOAP envelope.
Streaming	<p>Specifies whether streaming is supported for incoming and outgoing messages. Use the <code>TransferMode</code> property on the binding to set the value. The allowable values include:</p> <ul style="list-style-type: none"> - Buffered: The request and response messages are both buffered. - Streamed: The request and response messages are both streamed. - StreamedRequest: The request message is streamed and the response message is buffered. - StreamedResponse: The request message is buffered and the response message is streamed.

See also

[Endpoint Creation Overview](#)

[Using Bindings to Configure Services and Clients](#)

[Basic WCF Programming](#)

Using Bindings to Configure Services and Clients

5/5/2018 • 3 minutes to read • [Edit Online](#)

Bindings are objects that specify the communication details required to connect to an endpoint. More specifically, bindings contain configuration information that is used to create the client or service runtime by defining the specifics of transports, wire-formats (message encoding), and protocols to use for the respective endpoint or client channel. To create a functioning Windows Communication Foundation (WCF) service, each endpoint in the service requires a binding. This topic explains what bindings are, how they are defined, and how a particular binding is specified for an endpoint.

What a Binding Defines

The information in a binding can be very basic or very complex. The most basic binding specifies only the transport protocol (such as HTTP) that must be used to connect to the endpoint. More generally, the information a binding contains about how to connect to an endpoint falls into one of the categories in the following table.

Protocols

Determines the security mechanism being used, either reliable messaging capability or transaction context flow settings.

Transport

Determines the underlying transport protocol to use (for example, TCP or HTTP).

Encoding

Determines the message encoding, for example, text/XML, binary, or Message Transmission Optimization Mechanism (MTOM), which determines how messages are represented as byte streams on the wire.

System-Provided Bindings

WCF includes a set of system-provided bindings that are designed to cover most application requirements and scenarios. The following classes represent some examples of system-provided bindings:

- [BasicHttpBinding](#): An HTTP protocol binding suitable for connecting to Web services that conforms to the WS-I Basic Profile 1.1 specification (for example, ASP.NET Web services [ASMX]-based services).
- [WSHttpBinding](#): An HTTP protocol binding suitable for connecting to endpoints that conform to the Web services specifications protocols.
- [NetNamedPipeBinding](#): Uses the .NET binary encoding and framing technologies in conjunction with the Windows named pipe transport to connect to other WCF endpoints on the same machine.
- [NetMsmqBinding](#): Uses the .NET binary encoding and framing technologies in conjunction with the Message Queuing (also known as MSMQ) to create queued message connections with other WCF endpoints.

For a complete list of system-provided bindings, with descriptions, see [System-Provided Bindings](#).

Custom Bindings

If the system-provided binding collection does not have the correct combination of features that a service application requires, you can create a [CustomBinding](#) binding. For more information about the elements of a [CustomBinding](#) binding, see [<customBinding>](#) and [Custom Bindings](#).

Using Bindings

Using bindings entails two basic steps:

1. Select or define a binding. The easiest method is to choose one of the system-provided bindings and use its default settings. You can also choose a system-provided binding and reset its property values to suit your requirements. Alternatively, you can create a custom binding and set every property as required.
2. Create an endpoint that uses this binding.

Code and Configuration

You can define or configure bindings through code or configuration. These two approaches are independent of the type of binding used, for example, whether you are using a system-provided or a [CustomBinding](#) binding. In general, using code gives you complete control over the definition of a binding when you compile. Using configuration, on the other hand, allows a system administrator or the user of a WCF service or client to change the parameters of bindings. This flexibility is often desirable because there is no way to predict the specific machine requirements and network conditions into which a WCF application is to be deployed. Separating the binding (and addressing) information from the code allows administrators to change the binding details without having to recompile or redeploy the application. Note that if the binding is defined in code, it overwrites any configuration-based definitions made in the configuration file. For examples of these approaches, see the following topics:

- [How to: Host a WCF Service in a Managed Application](#) provides an example of creating a binding in code.
- [How to: Configure a Client](#) provides an example of creating a client using configuration.

See Also

[Endpoint Creation Overview](#)

[How to: Specify a Service Binding in Configuration](#)

[How to: Specify a Service Binding in Code](#)

[How to: Specify a Client Binding in Configuration](#)

[How to: Specify a Client Binding in Code](#)

How to: Specify a Client Binding in Code

5/4/2018 • 4 minutes to read • [Edit Online](#)

In this example, a client is created to use a calculator service and the binding for that client is specified imperatively in code. The client accesses the `CalculatorService`, which implements the `ICalculator` interface, and both the service and the client use the `BasicHttpBinding` class.

This procedure assumes that the calculator service is running. For information about building the service, see [How to: Specify a Service Binding in Configuration](#). It also uses the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) Windows Communication Foundation (WCF) provides to automatically generate the client components. The tool generates the client code for accessing the service.

The client is built in two parts. Svcutil.exe generates the `ClientCalculator` that implements the `ICalculator` interface. This client application is then constructed by constructing an instance of `ClientCalculator` and then specifying the binding and the address for the service in code.

For the source copy of this example, see the [BasicBinding](#) sample.

To specify a custom binding in code

1. Use Svcutil.exe from the command line to generate code from service metadata.

```
Svcutil.exe <service's Metadata Exchange (MEX) address or HTTP GET address>
```

2. The client that is generated contains the `ICalculator` interface that defines the service contract that the client implementation must satisfy.

```
[ServiceContract]
public interface ICalculator
{
    [OperationContract]
    double Add(double n1, double n2);
    [OperationContract]
    double Subtract(double n1, double n2);
    [OperationContract]
    double Multiply(double n1, double n2);
    [OperationContract]
    double Divide(double n1, double n2);
}
```



```
<ServiceContract(>> _  
Public Interface ICalculator  
  
    <OperationContract(>> _  
    Function Add(ByVal n1 As Double, ByVal n2 As Double) As Double  
  
    <OperationContract(>> _  
    Function Subtract(ByVal n1 As Double, ByVal n2 As Double) As Double  
  
    <OperationContract(>> _  
    Function Multiply(ByVal n1 As Double, ByVal n2 As Double) As Double  
  
    <OperationContract(>> _  
    Function Divide(ByVal n1 As Double, ByVal n2 As Double) As Double  
  
End Interface
```

3. The generated client also contains the implementation of the `ClientCalculator`.

```
public class CalculatorClient :
System.ServiceModel.ClientBase<Microsoft.ServiceModel.Samples.ICalculator>,
Microsoft.ServiceModel.Samples.ICalculator
{

    public CalculatorClient()
    {
    }

    public CalculatorClient(string endpointConfigurationName) :
        base(endpointConfigurationName)
    {
    }

    public CalculatorClient(string endpointConfigurationName, string remoteAddress) :
        base(endpointConfigurationName, remoteAddress)
    {
    }

    public CalculatorClient(string endpointConfigurationName, EndpointAddress remoteAddress) :
        base(endpointConfigurationName, remoteAddress)
    {
    }

    public CalculatorClient(Binding binding, EndpointAddress remoteAddress) :
        base(binding, remoteAddress)
    {
    }

    public double Add(double n1, double n2)
    {
        return base.Channel.Add(n1, n2);
    }

    public double Subtract(double n1, double n2)
    {
        return base.Channel.Subtract(n1, n2);
    }

    public double Multiply(double n1, double n2)
    {
        return base.Channel.Multiply(n1, n2);
    }

    public double Divide(double n1, double n2)
    {
        return base.Channel.Divide(n1, n2);
    }
}
```

```

Public Class CalculatorClient
    Inherits System.ServiceModel.ClientBase(Of Microsoft.ServiceModel.Samples.ICalculator)
    Implements Microsoft.ServiceModel.Samples.ICalculator

    Public Sub New()
    End Sub

    Public Sub New(ByVal endpointConfigurationName As String)
        MyBase.New(endpointConfigurationName)
    End Sub

    Public Sub New(ByVal endpointConfigurationName As String, _
        ByVal remoteAddress As String)
        MyBase.New(endpointConfigurationName, remoteAddress)
    End Sub

    Public Sub New(ByVal endpointConfigurationName As String, _
        ByVal remoteAddress As EndpointAddress)
        MyBase.New(endpointConfigurationName, remoteAddress)
    End Sub

    Public Sub New(ByVal binding As Binding, _
        ByVal remoteAddress As EndpointAddress)
        MyBase.New(binding, remoteAddress)
    End Sub

    Public Function Add(ByVal n1 As Double, _
        ByVal n2 As Double) As Double Implements
Microsoft.ServiceModel.Samples.ICalculator.Add
        Return MyBase.Channel.Add(n1, n2)
    End Function

    Public Function Subtract(ByVal n1 As Double, _
        ByVal n2 As Double) As Double Implements
Microsoft.ServiceModel.Samples.ICalculator.Subtract
        Return MyBase.Channel.Subtract(n1, n2)
    End Function

    Public Function Multiply(ByVal n1 As Double, _
        ByVal n2 As Double) As Double Implements
Microsoft.ServiceModel.Samples.ICalculator.Multiply
        Return MyBase.Channel.Multiply(n1, n2)
    End Function

    Public Function Divide(ByVal n1 As Double, _
        ByVal n2 As Double) As Double Implements
Microsoft.ServiceModel.Samples.ICalculator.Divide
        Return MyBase.Channel.Divide(n1, n2)
    End Function

End Class

```

4. Create an instance of the `ClientCalculator` that uses the [BasicHttpBinding](#) class in a client application, and then call the service operations at the specified address.

```

//Client implementation code.
class Client
{
    static void Main()
    {

        //Specify the binding to be used for the client.
        BasicHttpBinding binding = new BasicHttpBinding();

        //Specify the address to be used for the client.
        EndpointAddress address =
            new EndpointAddress("http://localhost/servicemodelsamples/service.svc");

        // Create a client that is configured with this address and binding.
        CalculatorClient client = new CalculatorClient(binding, address);

        // Call the Add service operation.
        double value1 = 100.00D;
        double value2 = 15.99D;
        double result = client.Add(value1, value2);
        Console.WriteLine("Add({0},{1}) = {2}", value1, value2, result);

        // Call the Subtract service operation.
        value1 = 145.00D;
        value2 = 76.54D;
        result = client.Subtract(value1, value2);
        Console.WriteLine("Subtract({0},{1}) = {2}", value1, value2, result);

        // Call the Multiply service operation.
        value1 = 9.00D;
        value2 = 81.25D;
        result = client.Multiply(value1, value2);
        Console.WriteLine("Multiply({0},{1}) = {2}", value1, value2, result);

        // Call the Divide service operation.
        value1 = 22.00D;
        value2 = 7.00D;
        result = client.Divide(value1, value2);
        Console.WriteLine("Divide({0},{1}) = {2}", value1, value2, result);

        //Closing the client gracefully closes the connection and cleans up resources
        client.Close();

        Console.WriteLine();
        Console.WriteLine("Press <ENTER> to terminate client.");
        Console.ReadLine();
    }
}

```

```

'Client implementation code.
Friend Class Client
    Shared Sub Main()

        'Specify the binding to be used for the client.
        Dim binding As New BasicHttpBinding()

        'Specify the address to be used for the client.
        Dim address As New EndpointAddress("http://localhost/servicemodelsamples/service.svc")

        ' Create a client that is configured with this address and binding.
        Dim client As New CalculatorClient(binding, address)

        ' Call the Add service operation.
        Dim value1 = 100.0R
        Dim value2 = 15.99R
        Dim result = client.Add(value1, value2)
        Console.WriteLine("Add({0},{1}) = {2}", value1, value2, result)

        ' Call the Subtract service operation.
        value1 = 145.0R
        value2 = 76.54R
        result = client.Subtract(value1, value2)
        Console.WriteLine("Subtract({0},{1}) = {2}", value1, value2, result)

        ' Call the Multiply service operation.
        value1 = 9.0R
        value2 = 81.25R
        result = client.Multiply(value1, value2)
        Console.WriteLine("Multiply({0},{1}) = {2}", value1, value2, result)

        ' Call the Divide service operation.
        value1 = 22.0R
        value2 = 7.0R
        result = client.Divide(value1, value2)
        Console.WriteLine("Divide({0},{1}) = {2}", value1, value2, result)

        'Closing the client gracefully closes the connection and cleans up resources
        client.Close()

        Console.WriteLine()
        Console.WriteLine("Press <ENTER> to terminate client.")
        Console.ReadLine()
    End Sub
End Class
End Namespace

```

5. Compile and run the client.

See Also

[Using Bindings to Configure Services and Clients](#)

How to: Specify a Client Binding in Configuration

5/4/2018 • 3 minutes to read • [Edit Online](#)

In this example, a client console application is created to use a calculator service, and the binding for that client is specified declaratively in configuration. The client accesses the `CalculatorService`, which implements the `ICalculator` interface, and both the service and the client use the `BasicHttpBinding` class.

The procedure outlined assumes that the calculator service is running. For information about how to build the service, see [How to: Specify a Service Binding in Configuration](#). It also uses the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) that Windows Communication Foundation (WCF) provides to automatically generate the client components. The tool generates the client code and configuration for accessing the service.

The client is built in two parts. Svcutil.exe generates the `ClientCalculator` that implements the `ICalculator` interface. This client application is then constructed by constructing an instance of `ClientCalculator`.

It is usually the best practice to specify the binding and address information declaratively in configuration rather than imperatively in code. Defining endpoints in code is usually not practical because the bindings and addresses for a deployed service are typically different from those used while the service is being developed. More generally, keeping the binding and addressing information out of the code allows them to change without having to recompile or redeploy the application.

You can perform all of the following configuration steps by using the [Configuration Editor Tool \(SvcConfigEditor.exe\)](#).

For the source copy of this example, see the [BasicBinding](#) sample.

Specifying a client binding in configuration

1. Use Svcutil.exe from the command line to generate code from service metadata.

```
Svcutil.exe <service's Metadata Exchange (MEX) address or HTTP GET address>
```

2. The client that is generated contains the `ICalculator` interface that defines the service contract that the client implementation must satisfy.

```

[System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")]
[System.ServiceModel.ServiceContractAttribute(Namespace="http://Microsoft.ServiceModel.Samples",
ConfigurationName="Microsoft.ServiceModel.Samples.ICalculator")]
public interface ICalculator
{

    [System.ServiceModel.OperationContractAttribute(Action="http://Microsoft.ServiceModel.Samples/ICalculator/Add", ReplyAction="http://Microsoft.ServiceModel.Samples/ICalculator/AddResponse")]
    double Add(double n1, double n2);

    [System.ServiceModel.OperationContractAttribute(Action="http://Microsoft.ServiceModel.Samples/ICalculator/Subtract", ReplyAction="http://Microsoft.ServiceModel.Samples/ICalculator/SubtractResponse")]
    double Subtract(double n1, double n2);

    [System.ServiceModel.OperationContractAttribute(Action="http://Microsoft.ServiceModel.Samples/ICalculator/Multiply", ReplyAction="http://Microsoft.ServiceModel.Samples/ICalculator/MultiplyResponse")]
    double Multiply(double n1, double n2);

    [System.ServiceModel.OperationContractAttribute(Action="http://Microsoft.ServiceModel.Samples/ICalculator/Divide", ReplyAction="http://Microsoft.ServiceModel.Samples/ICalculator/DivideResponse")]
    double Divide(double n1, double n2);
}

```

```

[ServiceContract]
public interface ICalculator
{
    [OperationContract]
    double Add(double n1, double n2);
    [OperationContract]
    double Subtract(double n1, double n2);
    [OperationContract]
    double Multiply(double n1, double n2);
    [OperationContract]
    double Divide(double n1, double n2);
}

```

3. The generated client also contains the implementation of the `ClientCalculator` .

```

[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")]
public partial class CalculatorClient :
    System.ServiceModel.ClientBase<Microsoft.ServiceModel.Samples.ICalculator>,
    Microsoft.ServiceModel.Samples.ICalculator
{
    public CalculatorClient()
    {
    }

    public CalculatorClient(string endpointConfigurationName) :
        base(endpointConfigurationName)
    {
    }

    public CalculatorClient(string endpointConfigurationName, string remoteAddress) :
        base(endpointConfigurationName, remoteAddress)
    {
    }

    public CalculatorClient(string endpointConfigurationName, System.ServiceModel.EndpointAddress
remoteAddress) :
        base(endpointConfigurationName, remoteAddress)
    {
    }

    public CalculatorClient(System.ServiceModel.Channels.Binding binding,
System.ServiceModel.EndpointAddress remoteAddress) :
        base(binding, remoteAddress)
    {
    }

    public double Add(double n1, double n2)
    {
        return base.Channel.Add(n1, n2);
    }

    public double Subtract(double n1, double n2)
    {
        return base.Channel.Subtract(n1, n2);
    }

    public double Multiply(double n1, double n2)
    {
        return base.Channel.Multiply(n1, n2);
    }

    public double Divide(double n1, double n2)
    {
        return base.Channel.Divide(n1, n2);
    }
}

```



```

public class CalculatorService : ICalculator
{
    public double Add(double n1, double n2)
    {
        return n1 + n2;
    }
    public double Subtract(double n1, double n2)
    {
        return n1 - n2;
    }
    public double Multiply(double n1, double n2)
    {
        return n1 * n2;
    }
    public double Divide(double n1, double n2)
    {
        return n1 / n2;
    }
}

```

4. Svcutil.exe also generates the configuration for the client that uses the [BasicHttpBinding](#) class. When using Visual Studio, name this file App.config. Note that the address and binding information are not specified anywhere inside the implementation of the service. Also, code does not have to be written to retrieve that information from the configuration file.

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>

    <client>
      <endpoint
        name=""
        address="http://localhost/servicemodelsamples/service.svc"
        binding="basicHttpBinding"
        contract="Microsoft.ServiceModel.Samples.ICalculator" />
    </client>

    <bindings>
      <basicHttpBinding/>
    </bindings>

  </system.serviceModel>

</configuration>

```

5. Create an instance of the `ClientCalculator` in an application, and then call the service operations.

```

using System;
using System.ServiceModel;

namespace Microsoft.ServiceModel.Samples
{
    //Client implementation code.
    class Client
    {
        static void Main()
        {
            // Create a client with given client endpoint configuration
            CalculatorClient client = new CalculatorClient();

            // Call the Add service operation.
            double value1 = 100.00D;
            double value2 = 15.99D;
            double result = client.Add(value1, value2);
            Console.WriteLine("Add({0},{1}) = {2}", value1, value2, result);

            // Call the Subtract service operation.
            value1 = 145.00D;
            value2 = 76.54D;
            result = client.Subtract(value1, value2);
            Console.WriteLine("Subtract({0},{1}) = {2}", value1, value2, result);

            // Call the Multiply service operation.
            value1 = 9.00D;
            value2 = 81.25D;
            result = client.Multiply(value1, value2);
            Console.WriteLine("Multiply({0},{1}) = {2}", value1, value2, result);

            // Call the Divide service operation.
            value1 = 22.00D;
            value2 = 7.00D;
            result = client.Divide(value1, value2);
            Console.WriteLine("Divide({0},{1}) = {2}", value1, value2, result);

            //Closing the client gracefully closes the connection and cleans up resources
            client.Close();

            Console.WriteLine();
            Console.WriteLine("Press <ENTER> to terminate client.");
            Console.ReadLine();
        }
    }
}

```

6. Compile and run the client.

See Also

[Using Bindings to Configure Services and Clients](#)

How to: Specify a Service Binding in Code

5/4/2018 • 3 minutes to read • [Edit Online](#)

In this example, an `ICalculator` contract is defined for a calculator service, the service is implemented in the `CalculatorService` class, and then its endpoint is defined in code, where it is specified that the service must use the `BasicHttpBinding` class.

It is usually the best practice to specify the binding and address information declaratively in configuration rather than imperatively in code. Defining endpoints in code is usually not practical because the bindings and addresses for a deployed service are typically different from those used while the service is being developed. More generally, keeping the binding and addressing information out of the code allows them to change without having to recompile or redeploy the application.

For a description of how to configure this service using configuration elements instead of code, see [How to: Specify a Service Binding in Configuration](#).

To specify in code to use the `BasicHttpBinding` for the service

1. Define a service contract for the type of service.

```
[ServiceContract]
public interface ICalculator
{
    [OperationContract]
    double Add(double n1, double n2);
    [OperationContract]
    double Subtract(double n1, double n2);
    [OperationContract]
    double Multiply(double n1, double n2);
    [OperationContract]
    double Divide(double n1, double n2);
}
```

```
<ServiceContract()> _
Public Interface ICalculator

    <OperationContract()> _
    Function Add(ByVal n1 As Double, _
                ByVal n2 As Double) As Double

    <OperationContract()> _
    Function Subtract(ByVal n1 As Double, ByVal _
                    n2 As Double) As Double

    <OperationContract()> _
    Function Multiply(ByVal n1 As Double, _
                    ByVal n2 As Double) As Double

    <OperationContract()> _
    Function Divide(ByVal n1 As Double, _
                   ByVal n2 As Double) As Double

End Interface
```

2. Implement the service contract in a service class.

```

public class CalculatorService : ICalculator
{
    public double Add(double n1, double n2)
    {
        return n1 + n2;
    }
    public double Subtract(double n1, double n2)
    {
        return n1 - n2;
    }
    public double Multiply(double n1, double n2)
    {
        return n1 * n2;
    }
    public double Divide(double n1, double n2)
    {
        return n1 / n2;
    }
}

```

```

Public Class CalculatorService
    Implements ICalculator

    Public Function Add(ByVal n1 As Double, _
                        ByVal n2 As Double) As Double Implements ICalculator.Add
        Return n1 + n2
    End Function

    Public Function Subtract(ByVal n1 As Double, _
                             ByVal n2 As Double) As Double Implements ICalculator.Subtract
        Return n1 - n2
    End Function

    Public Function Multiply(ByVal n1 As Double, _
                             ByVal n2 As Double) As Double Implements ICalculator.Multiply
        Return n1 * n2
    End Function

    Public Function Divide(ByVal n1 As Double, _
                           ByVal n2 As Double) As Double Implements ICalculator.Divide
        Return n1 / n2
    End Function

End Class

```

3. In the hosting application, create the base address for the service and the binding to use with the service.

```

// Specify a base address for the service

String baseAddress = "http://localhost/CalculatorService";
// Create the binding to be used by the service.

BasicHttpBinding binding1 = new BasicHttpBinding();

```

```
' Specify a base address for the service
Dim baseAddress = "http://localhost/CalculatorService"
' Create the binding to be used by the service.

Dim binding1 As New BasicHttpBinding()
```

4. Create the host for the service, add the endpoint, and then open the host.

```
using(ServiceHost host = new ServiceHost(typeof(CalculatorService)))
{
    host.AddServiceEndpoint(typeof(ICalculator),binding1, baseAddress);

    host.Open();
}
```

```
Using host As New ServiceHost(GetType(CalculatorService))
    With host
        .AddServiceEndpoint(GetType(ICalculator), _
                               binding1, _
                               baseAddress)

        host.Open()
    End With
End Using
```

To modify the default values of the binding properties

1. To modify one of the default property values of the [BasicHttpBinding](#) class, set the property value on the binding to the new value before creating the host. For example, to change the default open and close timeout values of 1 minute to 2 minutes, use the following.

```
TimeSpan modifiedCloseTimeout = new TimeSpan(00, 02, 00);
binding1.CloseTimeout = modifiedCloseTimeout;
```

```
Dim modifiedCloseTimeout As New TimeSpan(0, 2, 0)
binding1.CloseTimeout = modifiedCloseTimeout
```

See Also

[Using Bindings to Configure Services and Clients](#)
[Specifying an Endpoint Address](#)

How to: Specify a Service Binding in Configuration

5/4/2018 • 3 minutes to read • [Edit Online](#)

In this example, an `ICalculator` contract is defined for a basic calculator service, the service is implemented in the `CalculatorService` class, and then its endpoint is configured in the Web.config file, where it is specified that the service uses the [BasicHttpBinding](#). For a description of how to configure this service using code instead of a configuration, see [How to: Specify a Service Binding in Code](#).

It is usually the best practice to specify the binding and address information declaratively in configuration rather than imperatively in code. Defining endpoints in code is usually not practical because the bindings and addresses for a deployed service are typically different from those used while the service is being developed. More generally, keeping the binding and addressing information out of the code allows them to change without having to recompile or redeploy the application.

All of the following configuration steps can be undertaken using the [Configuration Editor Tool \(SvcConfigEditor.exe\)](#).

For the source copy of this example, see [BasicBinding](#).

To specify the BasicHttpBinding to use to configure the service

1. Define a service contract for the type of service.

```
[ServiceContract]
public interface ICalculator
{
    [OperationContract]
    double Add(double n1, double n2);
    [OperationContract]
    double Subtract(double n1, double n2);
    [OperationContract]
    double Multiply(double n1, double n2);
    [OperationContract]
    double Divide(double n1, double n2);
}
```

```
<ServiceContract(>> _
Public Interface ICalculator
    <OperationContract(>> _
    Function Add(ByVal n1 As Double, ByVal n2 As Double) As Double
    <OperationContract(>> _
    Function Subtract(ByVal n1 As Double, ByVal n2 As Double) As Double
    <OperationContract(>> _
    Function Multiply(ByVal n1 As Double, ByVal n2 As Double) As Double
    <OperationContract(>> _
    Function Divide(ByVal n1 As Double, ByVal n2 As Double) As Double
End Interface
```

2. Implement the service contract in a service class.

```

public class CalculatorService : ICalculator
{
    public double Add(double n1, double n2)
    {
        return n1 + n2;
    }
    public double Subtract(double n1, double n2)
    {
        return n1 - n2;
    }
    public double Multiply(double n1, double n2)
    {
        return n1 * n2;
    }
    public double Divide(double n1, double n2)
    {
        return n1 / n2;
    }
}

```

```

Public Class CalculatorService
    Implements ICalculator
    Public Function Add(ByVal n1 As Double, _
        ByVal n2 As Double) As Double Implements ICalculator.Add
        Return n1 + n2
    End Function
    Public Function Subtract(ByVal n1 As Double, _
        ByVal n2 As Double) As Double Implements ICalculator.Subtract
        Return n1 - n2
    End Function
    Public Function Multiply(ByVal n1 As Double, _
        ByVal n2 As Double) As Double Implements ICalculator.Multiply
        Return n1 * n2
    End Function
    Public Function Divide(ByVal n1 As Double, _
        ByVal n2 As Double) As Double Implements ICalculator.Divide
        Return n1 / n2
    End Function
End Class

```

NOTE

Address or binding information is not specified inside the implementation of the service. Also, code does not have to be written to fetch that information from the configuration file.

3. Create a Web.config file to configure an endpoint for the `CalculatorService` that uses the [WSHttpBinding](#).

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name=" CalculatorService" >
        <endpoint
          <-- Leave the address blank to be populated by default-->
          <--from the hosting environment,in this case IIS, so -->
          <-- the address will just be that of the IIS Virtual -->
          <--Directory.-->
          address=""
          <--Specify the binding type -->
          binding="wsHttpBinding"
          <--Specify the binding configuration name for that -->
          <--binding type. This is optional but useful if you -->
          <--want to modify the properties of the binding. -->
          <--The bindingConfiguration name Binding1 is defined -->
          <--below in the bindings element. -->
          bindingConfiguration="Binding1"
          contract="ICalculator" />
        </service>
      </services>
      <bindings>
        <wsHttpBinding>
          <binding name="Binding1">
            <-- Binding property values can be modified here. -->
            <--See the next procedure. -->
          </binding>
        </wsHttpBinding>
      </bindings>
    </system.serviceModel>
  </configuration>
```

4. Create a Service.svc file that contains the following line and place it in your Internet Information Services (IIS) virtual directory.

```
<%@ServiceHost language=c# Service="CalculatorService" %>
```

To modify the default values of the binding properties

1. To modify one of the default property values of the [WSHttpBinding](#), create a new binding configuration name - `<binding name="Binding1">` - within the `<wsHttpBinding>` element and set the new values for the attributes of the binding in this binding element. For example, to change the default open and close timeout values of 1 minute to 2 minutes, add the following to the configuration file.

```
<wsHttpBinding>
  <binding name="Binding1"
    closeTimeout="00:02:00"
    openTimeout="00:02:00">
  </binding>
</wsHttpBinding>
```

See Also

[Using Bindings to Configure Services and Clients](#)
[Specifying an Endpoint Address](#)

Configuring Bindings for Windows Communication Foundation Services

5/5/2018 • 4 minutes to read • [Edit Online](#)

When creating an application, you often want to defer decisions to the administrator after the deployment of the application. For example, there is often no way of knowing in advance what a service address, or Uniform Resource Identifier (URI), will be. Instead of hard-coding an address, it is preferable to allow an administrator to do so after creating a service. This flexibility is accomplished through configuration.

NOTE

Use the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) with the `/config` switch to quickly create configuration files.

Major Sections

The Windows Communication Foundation (WCF) configuration scheme includes the following three major sections (`system.serviceModel`, `bindings`, and `services`):

```
<configuration>
  <system.serviceModel>
    <bindings>
    </bindings>
    <services>
    </services>
    <behaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

ServiceModel Elements

You can use the section bounded by the `system.ServiceModel` element to configure a service type with one or more endpoints, as well as settings for a service. Each endpoint can then be configured with an address, a contract, and a binding. For more information about endpoints, see [Endpoint Creation Overview](#). If no endpoints are specified, the runtime adds default endpoints. For more information about default endpoints, bindings, and behaviors, see [Simplified Configuration](#) and [Simplified Configuration for WCF Services](#).

A binding specifies transports (HTTP, TCP, pipes, Message Queuing) and protocols (Security, Reliability, Transaction flows) and consists of binding elements, each of which specifies an aspect of how an endpoint communicates with the world.

For example, specifying the `<basicHttpBinding>` element indicates to use HTTP as the transport for an endpoint. This is used to wire up the endpoint at run time when the service using this endpoint is opened.

There are two kinds of bindings: predefined and custom. Predefined bindings contain useful combinations of elements that are used in common scenarios. For a list of predefined binding types that WCF provides, see [System-Provided Bindings](#). If no predefined binding collection has the correct combination of features that a service application needs, you can construct custom bindings to meet the application's requirements. For more information about custom bindings, see `<customBinding>`.

The following four examples illustrate the most common binding configurations used for setting up a WCF

service.

Specifying an Endpoint to Use a Binding Type

The first example illustrates how to specify an endpoint configured with an address, a contract, and a binding.

```
<service name="HelloWorld, IndigoConfig, Version=2.0.0.0, Culture=neutral, PublicKeyToken=null">
  <!-- This section is optional with the default configuration introduced
        in .NET Framework 4. -->
  <endpoint
    address="/HelloWorld2/"
    contract="HelloWorld, IndigoConfig, Version=2.0.0.0, Culture=neutral, PublicKeyToken=null"
    binding="basicHttpBinding" />
</service>
```

In this example, the `name` attribute indicates which service type the configuration is for. When you create a service in your code with the `HelloWorld` contract, it is initialized with all of the endpoints defined in the example configuration. If the assembly implements only one service contract, the `name` attribute can be omitted because the service uses the only available type. The attribute takes a string, which must be in the format

```
Namespace.Class, AssemblyName, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
```

The `address` attribute specifies the URI that other endpoints use to communicate to the service. The URI can be either an absolute or relative path. If a relative address is provided, the host is expected to provide a base address that is appropriate for the transport scheme used in the binding. If an address is not configured, the base address is assumed to be the address for that endpoint.

The `contract` attribute specifies the contract this endpoint is exposing. The service implementation type must implement the contract type. If a service implementation implements a single contract type, then this property can be omitted.

The `binding` attribute selects a predefined or custom binding to use for this specific endpoint. An endpoint that does not explicitly select a binding uses the default binding selection, which is `BasicHttpBinding`.

Modifying a Predefined Binding

In the following example, a predefined binding is modified. It can then be used to configure any endpoint in the service. The binding is modified by setting the `ReceiveTimeout` value to 1 second. Note that the property returns a `TimeSpan` object.

That altered binding is found in the bindings section. This altered binding can now be used when creating any endpoint by setting the `binding` attribute in the `endpoint` element.

NOTE

If you give a particular name to the binding, the `bindingConfiguration` specified in the service endpoint must match with it.

```
<service name="HelloWorld, IndigoConfig, Version=2.0.0.0, Culture=neutral, PublicKeyToken=null">
  <endpoint
    address="/HelloWorld2/"
    contract="HelloWorld, IndigoConfig, Version=2.0.0.0, Culture=neutral, PublicKeyToken=null"
    binding="basicHttpBinding" />
</service>
<bindings>
  <basicHttpBinding
    receiveTimeout="00:00:01"
  />
</bindings>
```

Configuring a Behavior to Apply to a Service

In the following example, a specific behavior is configured for the service type. The `ServiceMetadataBehavior` element is used to enable the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) to query the service and generate Web Services Description Language (WSDL) documents from the metadata.

NOTE

If you give a particular name to the behavior, the `behaviorConfiguration` specified in the service or endpoint section must match it.

```
<behaviors>
  <behavior>
    <ServiceMetadata httpGetEnabled="true" />
  </behavior>
</behaviors>
<services>
  <service
    name="HelloWorld, IndigoConfig, Version=2.0.0.0, Culture=neutral, PublicKeyToken=null">
    <endpoint
      address="http://computer:8080/Hello"
      contract="HelloWorld, IndigoConfig, Version=2.0.0.0, Culture=neutral, PublicKeyToken=null"
      binding="basicHttpBinding" />
    </service>
</services>
```

The preceding configuration enables a client to call and get the metadata of the "HelloWorld" typed service.

```
svcutil /config:Client.exe.config http://computer:8080/Hello?wsdl
```

Specifying a Service with Two Endpoints Using Different Binding Values

In this last example, two endpoints are configured for the `HelloWorld` service type. Each endpoint uses a different customized `bindingConfiguration` attribute of the same binding type (each modifies the `basicHttpBinding`).

```
<service name="HelloWorld, IndigoConfig, Version=2.0.0.0, Culture=neutral, PublicKeyToken=null">
  <endpoint
    address="http://computer:8080/Hello1"
    contract="HelloWorld, IndigoConfig, Version=2.0.0.0, Culture=neutral, PublicKeyToken=null"
    binding="basicHttpBinding"
    bindingConfiguration="shortTimeout" />
  <endpoint
    address="http://computer:8080/Hello2"
    contract="HelloWorld, IndigoConfig, Version=2.0.0.0, Culture=neutral, PublicKeyToken=null"
    binding="basicHttpBinding"
    bindingConfiguration="Secure" />
</service>
<bindings>
  <basicHttpBinding
    name="shortTimeout"
    timeout="00:00:00:01"
  />
  <basicHttpBinding
    name="Secure">
    <Security mode="Transport" />
  </basicHttpBinding>
</bindings>
```

You can get the same behavior using the default configuration by adding a `protocolMapping` section and configuring the bindings as demonstrated in the following example.

```
<protocolMapping>
  <add scheme="http" binding="basicHttpBinding" bindingConfiguration="shortTimeout" />
  <add scheme="https" binding="basicHttpBinding" bindingConfiguration="Secure" />
</protocolMapping>
<bindings>
  <basicHttpBinding
    name="shortTimeout"
    timeout="00:00:00:01"
  />
  <basicHttpBinding
    name="Secure" />
    <Security mode="Transport" />
</bindings>
```

See Also

[Simplified Configuration](#)

[System-Provided Bindings](#)

[Endpoint Creation Overview](#)

[Using Bindings to Configure Services and Clients](#)

Windows Communication Foundation Endpoints

5/5/2018 • 2 minutes to read • [Edit Online](#)

All communication with a Windows Communication Foundation (WCF) service occurs through the *endpoints* of the service. Endpoints provide clients access to the functionality that a WCF service offers.

For an overview about how to create an endpoint, see [Endpoint Creation Overview](#). Each endpoint contains:

- An address that indicates where to find the endpoint.
- A binding that specifies how a client can communicate with the endpoint.
- A contract that identifies the methods available.

For descriptions about how to specify these individual parts of an endpoint, see:

- [Specifying an Endpoint Address](#)
- [Using Bindings to Configure Services and Clients](#)
- [Designing and Implementing Services](#)

In This Section

[Endpoint Creation Overview](#)

Describes the structure of an endpoint and outlines how to define an endpoint in configuration and in code, as well as how to use the default endpoints, bindings, and behaviors provided by the runtime.

[Specifying an Endpoint Address](#)

Describes how communication with a WCF service occurs through endpoints.

[How to: Create a Service Endpoint in Configuration](#)

Demonstrates how to create a service endpoint in configuration.

[How to: Create a Service Endpoint in Code](#)

Demonstrates how to create a service endpoint in code.

[Publishing Metadata Endpoints](#)

Demonstrates how to publish metadata by publishing metadata endpoints in configuration and in code.

Reference

[EndpointAddress](#)

Related Sections

[Basic Programming Lifecycle](#)

Endpoint Creation Overview

10/3/2018 • 6 minutes to read • [Edit Online](#)

All communication with a Windows Communication Foundation (WCF) service occurs through the *endpoints* of the service. Endpoints provide the clients access to the functionality that a WCF service offers. This section describes the structure of an endpoint and outlines how to define an endpoint in configuration and in code.

The Structure of an Endpoint

Each endpoint contains an address that indicates where to find the endpoint, a binding that specifies how a client can communicate with the endpoint, and a contract that identifies the methods available.

- **Address.** The address uniquely identifies the endpoint and tells potential consumers where the service is located. It is represented in the WCF object model by the [EndpointAddress](#) address, which contains a Uniform Resource Identifier (URI) and address properties that include an identity, some Web Services Description Language (WSDL) elements, and a collection of optional headers. The optional headers provide additional detailed addressing information to identify or interact with the endpoint. For more information, see [Specifying an Endpoint Address](#).
- **Binding.** The binding specifies how to communicate with the endpoint. The binding specifies how the endpoint communicates with the world, including which transport protocol to use (for example, TCP or HTTP), which encoding to use for the messages (for example, text or binary), and which security requirements are necessary (for example, Secure Sockets Layer [SSL] or SOAP message security). For more information, see [Using Bindings to Configure Services and Clients](#).
- **Service contract.** The service contract outlines what functionality the endpoint exposes to the client. A contract specifies the operations that a client can call, the form of the message and the type of input parameters or data required to call the operation, and the kind of processing or response message the client can expect. Three basic types of contracts correspond to basic message exchange patterns (MEPs): datagram (one-way), request/reply, and duplex (bidirectional). The service contract can also employ data and message contracts to require specific data types and message formats when being accessed. For more information about how to define a service contract, see [Designing Service Contracts](#). Note that a client may also be required to implement a service-defined contract, called a callback contract, to receive messages from the service in a duplex MEP. For more information, see [Duplex Services](#).

The endpoint for a service can be specified either imperatively by using code or declaratively through configuration. If no endpoints are specified then the runtime provides default endpoints by adding one default endpoint for each base address for each service contract implemented by the service. Defining endpoints in code is usually not practical because the bindings and addresses for a deployed service are typically different from those used while the service is being developed. Generally, it is more practical to define service endpoints using configuration rather than code. Keeping the binding and addressing information out of the code allows them to change without having to recompile and redeploy the application.

NOTE

When adding a service endpoint that performs impersonation, you must either use one of the [AddServiceEndpoint](#) methods or the [GetContract\(Type, Type\)](#) method to properly load the contract into a new [ServiceDescription](#) object.

Defining Endpoints in Code

The following example illustrates how to specify an endpoint in code with the following:

- Define a contract for an `IEcho` type of service that accepts someone's name and echo with the response "Hello <name>!".
- Implement an `Echo` service of the type defined by the `IEcho` contract.
- Specify an endpoint address of `http://localhost:8000/Echo` for the service.
- Configure the `Echo` service using a [WSHttpBinding](#) binding.

```
Namespace Echo
{
    // Define the contract for the IEcho service
    [ServiceContract]
    public interface IEcho
    {
        [OperationContract]
        String Hello(string name)
    }

    // Create an Echo service that implements IEcho contract
    class Echo : IEcho
    {
        public string Hello(string name)
        {
            return "Hello" + name + "!";
        }
        public static void Main ()
        {
            //Specify the base address for Echo service.
            Uri echoUri = new Uri("http://localhost:8000/");

            //Create a ServiceHost for the Echo service.
            ServiceHost serviceHost = new ServiceHost(typeof(Echo),echoUri);

            // Use a predefined WSHttpBinding to configure the service.
            WSHttpBinding binding = new WSHttpBinding();

            // Add the endpoint for this service to the service host.
            serviceHost.AddServiceEndpoint(
                typeof(IEcho),
                binding,
                echoUri
            );

            // Open the service host to run it.
            serviceHost.Open();
        }
    }
}
```

```

' Define the contract for the IEcho service
<ServiceContract()> _
Public Interface IEcho
    <OperationContract()> _
    Function Hello(ByVal name As String) As String
End Interface

' Create an Echo service that implements IEcho contract
Public Class Echo
    Implements IEcho
    Public Function Hello(ByVal name As String) As String _
Implements ICalculator.Hello
        Dim result As String = "Hello" + name + "!"
        Return result
    End Function

' Specify the base address for Echo service.
Dim echoUri As Uri = New Uri("http://localhost:8000/")

' Create a ServiceHost for the Echo service.
Dim svcHost As ServiceHost = New ServiceHost(GetType>HelloWorld), echoUri)

' Use a predefined WSHttpBinding to configure the service.
Dim binding As New WSHttpBinding()

' Add the endpoint for this service to the service host.
serviceHost.AddServiceEndpoint(GetType(IEcho), binding, echoUri)

' Open the service host to run it.
serviceHost.Open()

```

NOTE

The service host is created with a base address and then the rest of the address, relative to the base address, is specified as part of an endpoint. This partitioning of the address allows multiple endpoints to be defined more conveniently for services at a host.

NOTE

Properties of [ServiceDescription](#) in the service application must not be modified subsequent to the [OnOpening](#) method on [ServiceHostBase](#). Some members, such as the [Credentials](#) property and the `AddServiceEndpoint` methods on [ServiceHostBase](#) and [ServiceHost](#), throw an exception if modified past that point. Others permit you to modify them, but the result is undefined.

Similarly, on the client the [ServiceEndpoint](#) values must not be modified after the call to [OnOpening](#) on the [ChannelFactory](#). The [Credentials](#) property throws an exception if modified past that point. The other client description values can be modified without error, but the result is undefined.

Whether for the service or client, it is recommended that you modify the description prior to calling [Open](#).

Defining Endpoints in Configuration

When creating an application, you often want to defer decisions to the administrator who is deploying the application. For example, there is often no way of knowing in advance what a service address (a URI) will be. Instead of hard-coding an address, it is preferable to allow an administrator to do so after creating a service. This flexibility is accomplished through configuration. For details, see [Configuring Services](#).

NOTE

Use the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) with the `/config: filename [, filename]` switch to quickly create configuration files.

Using Default Endpoints

If no endpoints are specified in code or in configuration then the runtime provides default endpoints by adding one default endpoint for each base address for each service contract implemented by the service. The base address can be specified in code or in configuration, and the default endpoints are added when [Open\(\)](#) is called on the [ServiceHost](#). This example is the same example from the previous section, but since no endpoints are specified, the default endpoints are added.

```
Namespace Echo
{
    // Define the contract for the IEcho service
    [ServiceContract]
    Interface IEcho
    {
        [OperationContract]
        String Hello(string name)
    }

    // Create an Echo service that implements IEcho contract
    Class Echo : IEcho
    {
        Public string Hello(string name)
        {
            return "Hello" + name + "!";
        }
        static void Main ()
        {
            //Specify the base address for Echo service.
            Uri echoUri = new Uri("http://localhost:8000/");

            //Create a ServiceHost for the Echo service.
            ServiceHost serviceHost = new ServiceHost(typeof(Echo),echoUri);

            // Open the service host to run it. Default endpoints
            // are added when the service is opened.
            serviceHost.Open();
        }
    }
}
```

```

' Define the contract for the IEcho service
<ServiceContract()> _
Public Interface IEcho
    <OperationContract()> _
    Function Hello(ByVal name As String) As String
End Interface

' Create an Echo service that implements IEcho contract
Public Class Echo
    Implements IEcho
    Public Function Hello(ByVal name As String) As String _
Implements ICalculator.Hello
        Dim result As String = "Hello" + name + "!"
        Return result
    End Function

' Specify the base address for Echo service.
Dim echoUri As Uri = New Uri("http://localhost:8000/")

' Open the service host to run it. Default endpoints
' are added when the service is opened.
serviceHost.Open()

```

If endpoints are explicitly provided, the default endpoints can still be added by calling [AddDefaultEndpoints](#) on the [ServiceHost](#) before calling [Open](#). For more information about default endpoints, see [Simplified Configuration](#) and [Simplified Configuration for WCF Services](#).

See Also

[Implementing Service Contracts](#)

Specifying an Endpoint Address

8/31/2018 • 7 minutes to read • [Edit Online](#)

All communication with a Windows Communication Foundation (WCF) service occurs through its endpoints. Each [ServiceEndpoint](#) contains an [Address](#), a [Binding](#), and a [Contract](#). The contract specifies which operations are available. The binding specifies how to communicate with the service, and the address specifies where to find the service. Every endpoint must have a unique address. The endpoint address is represented by the [EndpointAddress](#) class, which contains a Uniform Resource Identifier (URI) that represents the address of the service, an [Identity](#), which represents the security identity of the service, and a collection of optional [Headers](#). The optional headers provide more detailed addressing information to identify or interact with the endpoint. For example, headers can indicate how to process an incoming message, where the endpoint should send a reply message, or which instance of a service to use to process an incoming message from a particular user when multiple instances are available.

Definition of an Endpoint Address

In WCF, an [EndpointAddress](#) models an endpoint reference (EPR) as defined in the WS-Addressing standard.

The address URI for most transports has four parts. For example, this URI,

`http://www.fabrikam.com:322/mathservice.svc/secureEndpoint` has the following four parts:

- Scheme: http:
- Machine: `www.fabrikam.com`
- (Optional) Port: 322
- Path: `/mathservice.svc/secureEndpoint`

Part of the EPR model is that each endpoint reference can carry some reference parameters that add extra identifying information. In WCF, these reference parameters are modeled as instances of the [AddressHeader](#) class.

The endpoint address for a service can be specified either imperatively by using code or declaratively through configuration. Defining endpoints in code is usually not practical because the bindings and addresses for a deployed service are typically different from those used while the service is being developed. Generally, it is more practical to define service endpoints using configuration rather than code. Keeping the binding and addressing information out of the code allows them to change without having to recompile and redeploy the application. If no endpoints are specified in code or in configuration, then the runtime adds one default endpoint on each base address for each contract implemented by the service.

There are two ways to specify endpoint addresses for a service in WCF. You can specify an absolute address for each endpoint associated with the service or you can provide a base address for the [ServiceHost](#) of a service and then specify an address for each endpoint associated with this service that is defined relative to this base address. You can use each of these procedures to specify the endpoint addresses for a service in either configuration or code. If you do not specify a relative address, the service uses the base address. You can also have multiple base addresses for a service, but each service is allowed only one base address for each transport. If you have multiple endpoints, each of which is configured with a different binding, their addresses must be unique. Endpoints that use the same binding but different contracts can use the same address.

When hosting with IIS, you do not manage the [ServiceHost](#) instance yourself. The base address is always the address specified in the .svc file for the service when hosting in IIS. So you must use relative endpoint addresses

for IIS-hosted service endpoints. Supplying a fully-qualified endpoint address can lead to errors in the deployment of the service. For more information, see [Deploying an Internet Information Services-Hosted WCF Service](#).

Defining Endpoint Addresses in Configuration

To define an endpoint in a configuration file, use the `<endpoint>` element.

```
<configuration>
  <system.serviceModel>
    <services>
      <service name="UE.Samples.HelloService"
        behaviorConfiguration="HelloServiceBehavior">
        <endpoint address="/Address1"
          binding="basicHttpBinding"
          contract="UE.Samples.IHello"/>

        <endpoint address="mex"
          binding="mexHttpBinding"
          contract="IMetadataExchange" />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="HelloServiceBehavior">
          <serviceMetadata httpGetEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

When the [Open](#) method is called (that is, when the hosting application attempts to start the service), the system looks for a `<service>` element with a name attribute that specifies "UE.Samples.HelloService". If the `<service>` element is found, the system loads the specified class and creates endpoints using the endpoint definitions provided in the configuration file. This mechanism allows you to load and start a service with two lines of code while keeping binding and addressing information out of your code. The advantage of this approach is that these changes can be made without having to recompile or redeploy the application.

The optional headers are declared in a `<headers>`. The following is an example of the elements used to specify endpoints for a service in a configuration file that distinguishes between two headers: "Gold" clients from `http://tempuri1.org/` and "Standard" clients from `http://tempuri2.org/`. The client calling this service must have the appropriate `<headers>` in its configuration file.

```

<configuration>
  <system.serviceModel>
    <services>
      <service name="UE.Samples.HelloService"
        behaviorConfiguration="HelloServiceBehavior">
        <endpoint address="/Address1"
          binding="basicHttpBinding"
          contract="UE.Samples.IHello">

          <headers>
            <Member xmlns="http://tempuri1.org/">Gold</Member>
          </headers>
        </endpoint>
        <endpoint address="/Address2"
          binding="basicHttpBinding"
          contract="UE.Samples.IHello">
          <headers>
            <Member xmlns="http://tempuri2.org/">Silver</Member>
          </headers>
        </endpoint>

        <endpoint address="mex"
          binding="mexHttpBinding"
          contract="IMetadataExchange" />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="HelloServiceBehavior">
          <serviceMetadata httpGetEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>

```

Headers can also be set on individual messages instead of all messages on an endpoint (as shown previously). This is done by using [OperationContextScope](#) to create a new context in a client application to add a custom header to the outgoing message, as shown in the following example.

```

SampleServiceClient wcfClient = new SampleServiceClient(new InstanceContext(this));
try
{
    using (OperationContextScope scope = new OperationContextScope(wcfClient.InnerChannel))
    {
        MessageHeader header
            = MessageHeader.CreateHeader(
                "Service-Bound-CustomHeader",
                "http://Microsoft.WCF.Documentation",
                "Custom Happy Value."
            );
        OperationContext.Current.OutgoingMessageHeaders.Add(header);

        // Making calls.
        Console.WriteLine("Enter the greeting to send: ");
        string greeting = Console.ReadLine();

        //Console.ReadLine();
        header = MessageHeader.CreateHeader(
            "Service-Bound-OneWayHeader",
            "http://Microsoft.WCF.Documentation",
            "Different Happy Value."
        );
        OperationContext.Current.OutgoingMessageHeaders.Add(header);

        // One-way
        wcfClient.Push(greeting);
        this.wait.WaitOne();

        // Done with service.
        wcfClient.Close();
        Console.WriteLine("Done!");
        Console.ReadLine();
    }
}
catch (TimeoutException timeProblem)
{
    Console.WriteLine("The service operation timed out. " + timeProblem.Message);
    Console.ReadLine();
    wcfClient.Abort();
}
catch (CommunicationException commProblem)
{
    Console.WriteLine("There was a communication problem. " + commProblem.Message);
    Console.ReadLine();
    wcfClient.Abort();
}
}

```

```

Dim wcfClient As New SampleServiceClient(New InstanceContext(Me))
Try
    Using scope As New OperationContextScope(wcfClient.InnerChannel)
        Dim header As MessageHeader = MessageHeader.CreateHeader("Service-Bound-CustomHeader", _
            "http://Microsoft.WCF.Documentation", "Custom Happy Value.")
        OperationContext.Current.OutgoingMessageHeaders.Add(header)

        ' Making calls.
        Console.WriteLine("Enter the greeting to send: ")
        Dim greeting As String = Console.ReadLine()

        ' Console.ReadLine();
        header = MessageHeader.CreateHeader("Service-Bound-OneWayHeader", _
            "http://Microsoft.WCF.Documentation", "Different Happy
Value.")
        OperationContext.Current.OutgoingMessageHeaders.Add(header)

        ' One-way
        wcfClient.Push(greeting)
        Me.Wait.WaitOne()

        ' Done with service.
        wcfClient.Close()
        Console.WriteLine("Done!")
        Console.ReadLine()
    End Using
Catch timeProblem As TimeoutException
    Console.WriteLine("The service operation timed out. " & timeProblem.Message)
    Console.ReadLine()
    wcfClient.Abort()
Catch commProblem As CommunicationException
    Console.WriteLine("There was a communication problem. " & commProblem.Message)
    Console.ReadLine()
    wcfClient.Abort()
End Try

```

Endpoint Address in Metadata

An endpoint address is represented in Web Services Description Language (WSDL) as a WS-Addressing `EndpointReference` (EPR) element inside the corresponding endpoint's `wsdl:port` element. The EPR contains the endpoint's address as well as any address properties. Note that the EPR inside `wsdl:port` replaces `soap:Address` as seen in the following example.

Defining Endpoint Addresses in Code

An endpoint address can be created in code with the [EndpointAddress](#) class. The URI specified for the endpoint address can be a fully-qualified path or a path that is relative to the service's base address. The following code illustrates how to create an instance of the [EndpointAddress](#) class and add it to the [ServiceHost](#) instance that is hosting the service.

The following example demonstrates how to specify the full endpoint address in code.

```
Uri baseAddress = new Uri("http://localhost:8000/HelloService");
string address = "http://localhost:8000/HelloService/MyService";

using (ServiceHost serviceHost = new ServiceHost(typeof(HelloService), baseAddress))
{
    serviceHost.AddServiceEndpoint(typeof(Hello), new BasicHttpBinding(), address);
    serviceHost.Open();
    Console.WriteLine("Press <enter> to terminate service");
    Console.ReadLine();
    serviceHost.Close();
}
```

The following example demonstrates how to add a relative address ("MyService") to the base address of the service host.

```
Uri baseAddress = new Uri("http://localhost:8000/HelloService");

using (ServiceHost serviceHost = new ServiceHost(typeof(HelloService), baseAddress))
{
    serviceHost.AddServiceEndpoint(typeof(Hello), new BasicHttpBinding(), "MyService");
    serviceHost.Open();
    Console.WriteLine("Press <enter> to terminate service");
    Console.ReadLine();
    serviceHost.Close();
}
```

NOTE

Properties of the [ServiceDescription](#) in the service application must not be modified subsequent to the [OnOpening](#) method on [ServiceHostBase](#). Some members, such as the [Credentials](#) property and the `AddServiceEndpoint` methods on [ServiceHostBase](#) and [ServiceHost](#), throw an exception if modified after that point. Others permit you to modify them, but the result is undefined.

Similarly, on the client the [ServiceEndpoint](#) values must not be modified after the call to [OnOpening](#) on the [ChannelFactory](#). The [Credentials](#) property throws an exception if modified after that point. The other client description values can be modified without error, but the result is undefined.

Whether for the service or client, it is recommended that you modify the description prior to calling [Open](#).

Using Default Endpoints

If no endpoints are specified in code or in configuration then the runtime provides default endpoints by adding one default endpoint on each base address for each service contract implemented by the service. The base address can be specified in code or in configuration, and the default endpoints are added when [Open](#) is called on the [ServiceHost](#).

If endpoints are explicitly provided, the default endpoints can still be added by calling [AddDefaultEndpoints](#) on the [ServiceHost](#) before calling [Open](#). For more information about default endpoints, bindings, and behaviors, see [Simplified Configuration](#) and [Simplified Configuration for WCF Services](#).

See Also

[EndpointAddress](#)

[Service Identity and Authentication](#)

[Endpoint Creation Overview](#)

[Hosting](#)

Publishing Metadata Endpoints

5/5/2018 • 2 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) services publish metadata by publishing one or more metadata endpoints. Publishing service metadata makes the metadata available using standardized protocols, such as WS-MetadataExchange (MEX) and HTTP/GET requests. Metadata endpoints are similar to other service endpoints in that they have an address, a binding, and a contract, and they can be added to a service host through configuration or in code. To enable publishing metadata endpoints, you must add the [ServiceMetadataBehavior](#) service behavior to the service.

In This Section

[How to: Publish Metadata for a Service Using a Configuration File](#)

Demonstrates how to configure a WCF service to publish metadata so that clients can retrieve the metadata using a WS-MetadataExchange or an HTTP/GET request using the `?wsdl` query string.

[How to: Publish Metadata for a Service Using Code](#)

Demonstrates how to enable metadata publishing for a WCF service in code so that clients can retrieve the metadata using a WS-MetadataExchange or an HTTP/GET request using the `?wsdl` query string.

See Also

[Publishing Metadata](#)

Securing Services

10/24/2018 • 13 minutes to read • [Edit Online](#)

Security of a Windows Communication Foundation (WCF) service consists of two primary requirements: transfer security and authorization. (A third requirement, auditing of security events, is described in [Auditing](#).) In brief, transfer security includes authentication (verifying the identity of both the service and the client), confidentiality (message encryption), and integrity (digital signing to detect tampering). Authorization is the control of access to resources, for example, allowing only privileged users to read a file. Using features of WCF, the two primary requirements are easily implemented.

With the exception of the [BasicHttpBinding](#) class (or the `<basicHttpBinding>` element in configuration), transfer security is enabled by default for all of the predefined bindings. The topics in this section cover two basic scenarios: implementing transfer security and authorization on an intranet service hosted on Internet Information Services (IIS), and implementing transfer security and authorization on a service hosted on IIS.

NOTE

Windows XP Home does not support Windows authentication. Therefore, you should not run a service on that system.

Security Basics

Security relies on *credentials*. A credential proves that an entity is who it claims to be. (An *entity* can be a person, a software process, a company, or anything that can be authorized.) For example, a client of a service makes a *claim of identity*, and the credential proves that claim in some manner. In a typical scenario, an exchange of credentials occurs. First, a service makes a claim of its identity and proves it to the client with a credential. Conversely, the client makes a claim of identity and presents a credential to the service. If both parties trust the other's credentials, then a *secure context* can be established in which all messages are exchanged in confidentiality, and all messages are signed to protect their integrity. After the service establishes the client's identity, it can then match the claims in the credential to a *role* or *membership* in a group. In either case, using the role or the group to which the client belongs, the service *authorizes* the client to perform a limited set of operations based on the role or group privileges.

Windows Security Mechanisms

If the client and the service computer are both on a Windows domain that requires both to log on to the network, the credentials are provided by Windows infrastructure. In that case, the credentials are established when a computer user logs on to the network. Every user and every computer on the network must be validated as belonging to the trusted set of users and computers. On a Windows system, every such user and computer is also known as a *security principal*.

On a Windows domain backed by a *Kerberos* controller, the Kerberos controller uses a scheme based on granting tickets to each security principal. The tickets the controller grants are trusted by other ticket granters in the system. Whenever an entity tries to perform some operation or access a *resource* (such as a file or directory on a machine), the ticket is examined for its validity and, if it passes, the principal is granted another ticket for the operation. This method of granting tickets is more efficient than the alternative of trying to validate the principal for every operation.

An older, less-secure mechanism used on Windows domains is NT LAN Manager (NTLM). In cases where Kerberos cannot be used (typically outside of a Windows domain, such as in a workgroup), NTLM can be used as an alternative. NTLM is also available as a security option for IIS.

On a Windows system, authorization works by assigning each computer and user to a set of roles and groups. For example, every Windows computer must be set up and controlled by a person (or group of people) in the role of the *administrator*. Another role is that of the *user*, which has a much more constrained set of permissions. In addition to the role, users are assigned to groups. A group allows multiple users to perform in the same role. In practice, therefore, a Windows machine is administered by assigning users to groups. For example, several users can be assigned to the group of users of a computer, and a much more constrained set of users can be assigned to the group of administrators. On a local machine, an administrator can also create new groups and assign other users (or even other groups) to the group.

On a computer running Windows, every folder in a directory can be protected. That is, you can select a folder and control who can access the files, and whether or not they can copy the file, or (in the most privileged case) change a file, delete a file, or add files to the folder. This is known as access control, and the mechanism for it is known as the access control list (ACL). When creating the ACL, you can assign access privileges to any group or groups, as well as individual members of a domain.

The WCF infrastructure is designed to use these Windows security mechanisms. Therefore, if you are creating a service that is deployed on an intranet, and whose clients are restricted to members of the Windows domain, security is easily implemented. Only valid users can log on to the domain. After users are logged on, the Kerberos controller enables each user to establish secure contexts with any other computer or application. On a local machine, groups can be easily created, and when protecting specific folders, those groups can be used to assign access privileges on the computer.

Implementing Windows Security on Intranet Services

To secure an application that runs exclusively on a Windows domain, you can use the default security settings of either the [WSHttpBinding](#) or the [NetTcpBinding](#) binding. By default, anyone on the same Windows domain can access WCF services. Because those users have logged on to the network, they are trusted. The messages between a service and a client are encrypted for confidentiality and signed for integrity. For more information about how to create a service that uses Windows security, see [How to: Secure a Service with Windows Credentials](#).

Authorization Using the `PrincipalPermissionAttribute` Class

If you need to restrict the access of resources on a computer, the easiest way is to use the [PrincipalPermissionAttribute](#) class. This attribute enables you to restrict the invocation of service operations by demanding that the user be in a specified Windows group or role, or to be a specific user. For more information, see [How to: Restrict Access with the PrincipalPermissionAttribute Class](#).

Impersonation

Impersonation is another mechanism that you can use to control access to resources. By default, a service hosted by IIS will run under the identity of the ASPNET account. The ASPNET account can access only the resources for which it has permission. However, it is possible to set the ACL for a folder to exclude the ASPNET service account, but allow certain other identities to access the folder. The question then becomes how to allow those users to access the folder if the ASPNET account is not allowed to do so. The answer is to use impersonation, whereby the service is allowed to use the credentials of the client to access a particular resource. Another example is when accessing a SQL Server database to which only certain users have permission. For more information about using impersonation, see [How to: Impersonate a Client on a Service](#) and [Delegation and Impersonation](#).

Security on the Internet

Security on the Internet consists of the same requirements as security on an intranet. A service needs to present its credentials to prove its authenticity, and clients need to prove their identity to the service. Once a client's identity is proven, the service can control how much access the client has to resources. However, due to the heterogeneous nature of the Internet, the credentials presented differ from those used on a Windows domain.

Whereas a Kerberos controller handles the authentication of users on a domain with tickets for credentials, on the Internet, services and clients rely on any one of several different ways to present credentials. The objective of this topic, however, is to present a common approach that enables you to create a WCF service that is accessible on the Internet.

Using IIS and ASP.NET

The requirements of Internet security, and the mechanisms to solve those problems, are not new. IIS is Microsoft's Web server for the Internet and has many security features that address those problems; in addition, ASP.NET includes security features that WCF services can use. To take advantage of these security features, host an WCF service on IIS.

Using ASP.NET Membership and Role Providers

ASP.NET includes a membership and role provider. The provider is a database of user name/password pairs for authenticating callers that also allows you to specify each caller's access privileges. With WCF, you can easily use a pre-existing membership and role provider through configuration. For a sample application that demonstrates this, see the [Membership and Role Provider](#) sample.

Credentials Used by IIS

Unlike a Windows domain backed by a Kerberos controller, the Internet is an environment without a single controller to manage the millions of users logging on at any time. Instead, credentials on the Internet most often are in the form of X.509 certificates (also known as Secure Sockets Layer, or SSL, certificates). These certificates are typically issued by a *certification authority*, which can be a third-party company that vouches for the authenticity of the certificate and the person it has been issued to. To expose your service on the Internet, you must also supply such a trusted certificate to authenticate your service.

The question arises at this point, how do you get such a certificate? One approach is to go to a third-party certification authority, such as Authenticode or VeriSign, when you are ready to deploy your service, and purchase a certificate for your service. However, if you are in the development phase with WCF and not yet ready to commit to purchasing a certificate, tools and techniques exist for creating X.509 certificates that you can use to simulate a production deployment. For more information, see [Working with Certificates](#).

Security Modes

Programming WCF security entails a few critical decision points. One of the most basic is the choice of *security mode*. The two major security modes are *transport mode* and *message mode*.

A third mode, which combines the semantics of both major modes, is *transport with message credentials mode*.

The security mode determines how messages are secured, and each choice has advantages and disadvantages, as explained below. For more information about setting the security mode, see [How to: Set the Security Mode](#).

Transport Mode

There are several layers between the network and the application. One of these is the *transport* layer^{*,*} which manages the transfer of messages between endpoints. For the present purpose, it is only required that you understand that WCF uses several transport protocols, each of which can secure the transfer of messages. (For more information about transports, see [Transports](#).)

A commonly used protocol is HTTP; another is TCP. Each of these protocols can secure message transfer by a mechanism (or mechanisms) particular to the protocol. For example, the HTTP protocol is secured using SSL over HTTP, commonly abbreviated as "HTTPS." Thus, when you select the transport mode for security, you are choosing to use the mechanism as dictated by the protocol. For example, if you select the [WSHttpBinding](#) class and set its security mode to Transport, you are selecting SSL over HTTP (HTTPS) as the security mechanism. The advantage of the transport mode is that it is more efficient than message mode because security is integrated at a comparatively low level. When using transport mode, the security mechanism must be implemented according to the specification for the transport, and thus messages can flow securely only from point-to-point over the transport.

Message Mode

In contrast, message mode provides security by including the security data as part of every message. Using XML and SOAP security headers, the credentials and other data needed to ensure the integrity and confidentiality of the message are included with every message. Every message includes security data, so it results in a toll on performance because each message must be individually processed. (In transport mode, once the transport layer is secured, all messages flow freely.) However, message security has one advantage over transport security: it is more flexible. That is, the security requirements are not determined by the transport. You can use any type of client credential to secure the message. (In transport mode, the transport protocol determines the type of client credential that you can use.)

Transport with Message Credentials

The third mode combines the best of both transport and message security. In this mode, transport security is used to efficiently ensure the confidentiality and integrity of every message. At the same time, every message includes its credential data, which allows the message to be authenticated. With authentication, authorization can also be implemented. By authenticating a sender, access to resources can be granted (or denied) according to the sender's identity.

Specifying the Client Credential Type and Credential Value

After you have selected a security mode, you may also want to specify a client credential type. The client credential type specifies what type a client must use to authenticate itself to the server.

Not all scenarios require a client credential type, however. Using SSL over HTTP (HTTPS), a service authenticates itself to the client. As part of this authentication, the service's certificate is sent to the client in a process called *negotiation*. The SSL-secured transport ensures that all messages are confidential.

If you are creating a service that requires that the client be authenticated, your choice of a client credential type depends on the transport and mode selected. For example, using the HTTP transport and choosing transport mode gives you several choices, such as Basic, Digest, and others. (For more information about these credential types, see [Understanding HTTP Authentication](#).)

If you are creating a service on a Windows domain that will be available only to other users of the network, the easiest to use is the Windows client credential type. However, you may also need to provide a service with a certificate. This is shown in [How to: Specify Client Credential Values](#).

Credential Values

A *credential value* is the actual credential the service uses. Once you have specified a credential type, you may also need to configure your service with the actual credentials. If you have selected Windows (and the service will run on a Windows domain), then you do not specify an actual credential value.

Identity

In WCF, the term *identity* has different meanings to the server and the client. In brief, when running a service, an identity is assigned to the security context after authentication. To view the actual identity, check the [WindowsIdentity](#) and [PrimaryIdentity](#) properties of the [ServiceSecurityContext](#) class. For more information, see [How to: Examine the Security Context](#).

In contrast, on the client, identity is used to validate the service. At design time, a client developer can set the `<identity>` element to a value obtained from the service. At run time, the client checks the value of the element against the actual identity of the service. If the check fails, the client terminates the communication. The value can be a user principal name (UPN) if the service runs under a particular user's identity or a service principal name (SPN) if the service runs under a machine account. For more information, see [Service Identity and Authentication](#). The credential could also be a certificate, or a field found on a certificate that identifies the certificate.

Protection Levels

The `ProtectionLevel` property occurs on several attribute classes (such as the [ServiceContractAttribute](#) and the [OperationContractAttribute](#) classes). The protection level is a value that specifies whether the messages (or message parts) that support a service are signed, signed and encrypted, or sent without signatures or encryption. For more information about the property, see [Understanding Protection Level](#), and for programming examples, see [How to: Set the ProtectionLevel Property](#). For more information about designing a service contract with the `ProtectionLevel` in context, see [Designing Service Contracts](#).

See Also

- [System.ServiceModel](#)
- [ServiceCredentials](#)
- [ServiceContractAttribute](#)
- [OperationContractAttribute](#)
- [Service Identity and Authentication](#)
- [Understanding Protection Level](#)
- [Delegation and Impersonation](#)
- [Designing Service Contracts](#)
- [Security](#)
- [Security Overview](#)
- [How to: Set the ProtectionLevel Property](#)
- [How to: Secure a Service with Windows Credentials](#)
- [How to: Set the Security Mode](#)
- [How to: Specify the Client Credential Type](#)
- [How to: Restrict Access with the PrincipalPermissionAttribute Class](#)
- [How to: Impersonate a Client on a Service](#)
- [How to: Examine the Security Context](#)

How to: Secure a Service with Windows Credentials

10/24/2018 • 11 minutes to read • [Edit Online](#)

This topic shows how to enable transport security on a Windows Communication Foundation (WCF) service that resides in a Windows domain and is called by clients in the same domain. For more information about this scenario, see [Transport Security with Windows Authentication](#). For a sample application, see the [WSHttpBinding](#) sample.

This topic assumes you have an existing contract interface and implementation already defined, and adds on to that. You can also modify an existing service and client.

You can secure a service with Windows credentials completely in code. Alternatively, you can omit some of the code by using a configuration file. This topic shows both ways. Be sure you only use one of the ways, not both.

The first three procedures show how to secure the service using code. The fourth and fifth procedure shows how to do it with a configuration file.

Using Code

The complete code for the service and the client is in the Example section at the end of this topic.

The first procedure walks through creating and configuring a [WSHttpBinding](#) class in code. The binding uses the HTTP transport. The same binding is used on the client.

To create a [WSHttpBinding](#) that uses Windows credentials and message security

1. This procedure's code is inserted at the beginning of the `Run` method of the `Test` class in the service code in the Example section.
2. Create an instance of the [WSHttpBinding](#) class.
3. Set the `Mode` property of the [WSHttpSecurity](#) class to `Message`.
4. Set the `ClientCredentialType` property of the [MessageSecurityOverHttp](#) class to `Windows`.
5. The code for this procedure is as follows:

```
// First procedure:  
// create a WSHttpBinding that uses Windows credentials and message security  
WSHttpBinding myBinding = new WSHttpBinding();  
myBinding.Security.Mode = SecurityMode.Message;  
myBinding.Security.Message.ClientCredentialType =  
    MessageCredentialType.Windows;
```

```
Dim myBinding As New WSHttpBinding()  
myBinding.Security.Mode = SecurityMode.Message  
myBinding.Security.Message.ClientCredentialType = MessageCredentialType.Windows
```

Using the Binding in a Service

This is the second procedure, which shows how to use the binding in a self-hosted service. For more information about hosting services see [Hosting Services](#).

To use a binding in a service

1. Insert this procedure's code after the code from the preceding procedure.

2. Create a [Type](#) variable named `contractType` and assign it the type of the interface (`ICalculator`). When using Visual Basic, use the `GetType` operator; when using C#, use the `typeof` keyword.
3. Create a second `Type` variable named `serviceType` and assign it the type of the implemented contract (`Calculator`).
4. Create an instance of the [Uri](#) class named `baseAddress` with the base address of the service. The base address must have a scheme that matches the transport. In this case, the transport scheme is HTTP, and the address includes the special Uniform Resource Identifier (URI) "localhost" and a port number (8036) as well as a base endpoint address ("serviceModelSamples/"): `http://localhost:8036/serviceModelSamples/`.
5. Create an instance of the [ServiceHost](#) class with the `serviceType` and `baseAddress` variables.
6. Add an endpoint to the service using the `contractType`, binding, and an endpoint name (secureCalculator). A client must concatenate the base address and the endpoint name when initiating a call to the service.
7. Call the [Open](#) method to start the service. The code for this procedure is shown here:

```
// 2nd Procedure:
// Use the binding in a service
// Create the Type instances for later use and the URI for
// the base address.
Type contractType = typeof(ICalculator);
Type serviceType = typeof(Calculator);
Uri baseAddress = new
    Uri("http://localhost:8036/SecuritySamples/");

// Create the ServiceHost and add an endpoint, then start
// the service.
ServiceHost myServiceHost =
    new ServiceHost(serviceType, baseAddress);
myServiceHost.AddServiceEndpoint
    (contractType, myBinding, "secureCalculator");

//enable metadata
ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
smb.HttpGetEnabled = true;
myServiceHost.Description.Behaviors.Add(smb);

myServiceHost.Open();
```

```
' Create the Type instances for later use and the URI for
' the base address.
Dim contractType As Type = GetType(ICalculator)
Dim serviceType As Type = GetType(Calculator)
Dim baseAddress As New Uri("http://localhost:8036/serviceModelSamples/")

' Create the ServiceHost and add an endpoint, then start
' the service.
Dim myServiceHost As New ServiceHost(serviceType, baseAddress)
myServiceHost.AddServiceEndpoint(contractType, myBinding, "secureCalculator")
myServiceHost.Open()
```

Using the Binding in a Client

This procedure shows how to generate a proxy that communicates with the service. The proxy is generated with the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) which uses the service metadata to create the proxy.

This procedure also creates an instance of the [WSHttpBinding](#) class to communicate with the service, and then calls the service.

This example uses only code to create the client. As an alternative, you can use a configuration file, which is shown

in the section following this procedure.

To use a binding in a client with code

1. Use the SvcUtil.exe tool to generate the proxy code from the service's metadata. For more information, see [How to: Create a Client](#). The generated proxy code inherits from the `ClientBase<TChannel>` class, which ensures that every client has the necessary constructors, methods, and properties to communicate with a WCF service. In this example, the generated code includes the `CalculatorClient` class, which implements the `ICalculator` interface, enabling compatibility with the service code.
2. This procedure's code is inserted at the beginning of the `Main` method of the client program.
3. Create an instance of the `WSHttpBinding` class and set its security mode to `Message` and its client credential type to `Windows`. The example names the variable `clientBinding`.
4. Create an instance of the `EndpointAddress` class named `serviceAddress`. Initialize the instance with the base address concatenated with the endpoint name.
5. Create an instance of the generated client class with the `serviceAddress` and the `clientBinding` variables.
6. Call the `Open` method, as shown in the following code.
7. Call the service and display the results.

```
// 3rd Procedure:
// Creating a binding and using it in a service

// To run using config, comment the following lines, and uncomment out the code
// following this section
WSHttpBinding b = new WSHttpBinding(SecurityMode.Message);
b.Security.Message.ClientCredentialType = MessageCredentialType.Windows;

EndpointAddress ea = new EndpointAddress("Http://localhost:8036/SecuritySamples/secureCalculator");
CalculatorClient cc = new CalculatorClient(b, ea);
cc.Open();

// Now call the service and display the results
// Call the Add service operation.
double value1 = 100.00D;
double value2 = 15.99D;
double result = cc.Add(value1, value2);
Console.WriteLine("Add({0},{1}) = {2}", value1, value2, result);

// Closing the client gracefully closes the connection and cleans up resources.
cc.Close();
```

```
Dim b As New WSHttpBinding(SecurityMode.Message)
b.Security.Message.ClientCredentialType = MessageCredentialType.Windows

Dim ea As New EndpointAddress("net.tcp://machinename:8036/endpoint")
Dim cc As New CalculatorClient(b, ea)
cc.Open()

' Alternatively, use a binding name from a configuration file generated by the
' SvcUtil.exe tool to create the client. Omit the binding and endpoint address
' because that information is provided by the configuration file.
' CalculatorClass cc = new CalculatorClient("ICalculator_Binding")
```

Using the Configuration File

Instead of creating the binding with procedural code, you can use the following code shown for the bindings section of the configuration file.

If you do not already have a service defined, see [Designing and Implementing Services](#), and [Configuring Services](#).

Note This configuration code is used in both the service and client configuration files.

To enable transfer security on a service in a Windows domain using configuration

1. Add a `<wsHttpBinding>` element to the `<bindings>` element section of the configuration file.
2. Add a `<binding>` element to the `<wsHttpBinding>` element and set the `configurationName` attribute to a value appropriate to your application.
3. Add a `<security>` element and set the `mode` attribute to Message.
4. Add a `<message>` element and set the `clientCredentialType` attribute to Windows.
5. In the service's configuration file, replace the `<bindings>` section with the following code. If you do not already have a service configuration file, see [Using Bindings to Configure Services and Clients](#).

```
<bindings>
  <wsHttpBinding>
    <binding name = "wsHttpBinding_Calculator">
      <security mode="Message">
        <message clientCredentialType="Windows"/>
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
```

Using the Binding in a Client

This procedure shows how to generate two files: a proxy that communicates with the service and a configuration file. It also describes changes to the client program, which is the third file used on the client.

To use a binding in a client with configuration

1. Use the SvcUtil.exe tool to generate the proxy code and configuration file from the service's metadata. For more information, see [How to: Create a Client](#).
2. Replace the `<bindings>` section of the generated configuration file with the configuration code from the preceding section.
3. Procedural code is inserted at the beginning of the `Main` method of the client program.
4. Create an instance of the generated client class passing the name of the binding in the configuration file as an input parameter.
5. Call the `Open` method, as shown in the following code.
6. Call the service and display the results.

```
// 4th Procedure:
// Using config instead of the binding-related code
// In this case, use a binding name from a configuration file generated by the
// SvcUtil.exe tool to create the client. Omit the binding and endpoint address
// because that information is provided by the configuration file.

CalculatorClient cc = new CalculatorClient("ICalculator_Binding");
cc.Open();

// Now call the service and display the results
// Call the Add service operation.
double value1 = 100.00D;
double value2 = 15.99D;
double result = cc.Add(value1, value2);
Console.WriteLine("Add({0},{1}) = {2}", value1, value2, result);

// Closing the client gracefully closes the connection and cleans up resources.
cc.Close();
```

Example

```
using System;
using System.Collections;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Description;
using System.Security.Permissions;

[assembly: SecurityPermission(SecurityAction.RequestMinimum, Execution = true)]
namespace Microsoft.Security.Samples
{
    public class Test
    {
        static void Main()
        {
            Test t = new Test();
            Console.WriteLine("Starting...");
            t.Run();
        }

        private void Run()
        {
            // First procedure:
            // create a WSHttpBinding that uses Windows credentials and message security
            WSHttpBinding myBinding = new WSHttpBinding();
            myBinding.Security.Mode = SecurityMode.Message;
            myBinding.Security.Message.ClientCredentialType =
                MessageCredentialType.Windows;

            // 2nd Procedure:
            // Use the binding in a service
            // Create the Type instances for later use and the URI for
            // the base address.
            Type contractType = typeof(ICalculator);
            Type serviceType = typeof(Calculator);
            Uri baseAddress = new
                Uri("http://localhost:8036/SecuritySamples/");

            // Create the ServiceHost and add an endpoint, then start
            // the service.
            ServiceHost myServiceHost =
                new ServiceHost(serviceType, baseAddress);
            myServiceHost.AddServiceEndpoint
                (contractType, myBinding, "secureCalculator");
```

```

        //enable metadata
        ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
        smb.HttpGetEnabled = true;
        myServiceHost.Description.Behaviors.Add(smb);

        myServiceHost.Open();
        Console.WriteLine("Listening");
        Console.WriteLine("Press Enter to close the service");
        Console.ReadLine();
        myServiceHost.Close();
    }
}

[ServiceContract]
public interface ICalculator
{
    [OperationContract]
    double Add(double a, double b);
}

public class Calculator : ICalculator
{
    public double Add(double a, double b)
    {
        return a + b;
    }
}
}

```

```

using System;
using System.Collections.Generic;
using System.ServiceModel;

namespace Client
{
    static class SecureClientCode
    {
        static void Main()
        {
            // 3rd Procedure:
            // Creating a binding and using it in a service

            // To run using config, comment the following lines, and uncomment out the code
            // following this section
            WSHttpBinding b = new WSHttpBinding(SecurityMode.Message);
            b.Security.Message.ClientCredentialType = MessageCredentialType.Windows;

            EndpointAddress ea = new
            EndpointAddress("Http://localhost:8036/SecuritySamples/secureCalculator");
            CalculatorClient cc = new CalculatorClient(b, ea);
            cc.Open();

            // Now call the service and display the results
            // Call the Add service operation.
            double value1 = 100.00D;
            double value2 = 15.99D;
            double result = cc.Add(value1, value2);
            Console.WriteLine("Add({0},{1}) = {2}", value1, value2, result);

            // Closing the client gracefully closes the connection and cleans up resources.
            cc.Close();
        }

        static void Main2()
        {
            // 4th Procedure:

```

```

        // Using config instead of the binding-related code
        // In this case, use a binding name from a configuration file generated by the
        // SvcUtil.exe tool to create the client. Omit the binding and endpoint address
        // because that information is provided by the configuration file.

        CalculatorClient cc = new CalculatorClient("ICalculator_Binding");
        cc.Open();

        // Now call the service and display the results
        // Call the Add service operation.
        double value1 = 100.00D;
        double value2 = 15.99D;
        double result = cc.Add(value1, value2);
        Console.WriteLine("Add({0},{1}) = {2}", value1, value2, result);

        // Closing the client gracefully closes the connection and cleans up resources.
        cc.Close();
    }
}

```

```

[System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")]
[System.ServiceModel.ServiceContractAttribute(Namespace = "http://Microsoft.ServiceModel.Samples",
ConfigurationName = "ICalculator")]
public interface ICalculator
{

    [System.ServiceModel.OperationContractAttribute(Action =
"http://Microsoft.ServiceModel.Samples/ICalculator/Add", ReplyAction =
"http://Microsoft.ServiceModel.Samples/ICalculator/AddResponse")]
    double Add(double n1, double n2);

    [System.ServiceModel.OperationContractAttribute(Action =
"http://Microsoft.ServiceModel.Samples/ICalculator/Subtract", ReplyAction =
"http://Microsoft.ServiceModel.Samples/ICalculator/SubtractResponse")]
    double Subtract(double n1, double n2);

    [System.ServiceModel.OperationContractAttribute(Action =
"http://Microsoft.ServiceModel.Samples/ICalculator/Multiply", ReplyAction =
"http://Microsoft.ServiceModel.Samples/ICalculator/MultiplyResponse")]
    double Multiply(double n1, double n2);

    [System.ServiceModel.OperationContractAttribute(Action =
"http://Microsoft.ServiceModel.Samples/ICalculator/Divide", ReplyAction =
"http://Microsoft.ServiceModel.Samples/ICalculator/DivideResponse")]
    double Divide(double n1, double n2);
}

[System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")]
public interface ICalculatorChannel : ICalculator, System.ServiceModel.IClientChannel
{
}

[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")]
public class CalculatorClient : System.ServiceModel.ClientBase<ICalculator>, ICalculator
{

    public CalculatorClient()
    {
    }

    public CalculatorClient(string endpointConfigurationName)
        :
            base(endpointConfigurationName)
    {
    }

    public CalculatorClient(string endpointConfigurationName, string remoteAddress)

```

```

        :
        base(endpointConfigurationName, remoteAddress)
    {
    }

    public CalculatorClient(string endpointConfigurationName, System.ServiceModel.EndpointAddress
remoteAddress)
    :
        base(endpointConfigurationName, remoteAddress)
    {
    }

    public CalculatorClient(System.ServiceModel.Channels.Binding binding,
System.ServiceModel.EndpointAddress remoteAddress)
    :
        base(binding, remoteAddress)
    {
    }

    public double Add(double n1, double n2)
    {
        return base.Channel.Add(n1, n2);
    }

    public double Subtract(double n1, double n2)
    {
        return base.Channel.Subtract(n1, n2);
    }

    public double Multiply(double n1, double n2)
    {
        return base.Channel.Multiply(n1, n2);
    }

    public double Divide(double n1, double n2)
    {
        return base.Channel.Divide(n1, n2);
    }
}
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.ServiceModel

```

```

Public Class Program

```

```

    Shared Sub Main()

```

```

        Dim b As New WSHttpBinding(SecurityMode.Message)
        b.Security.Message.ClientCredentialType = MessageCredentialType.Windows

```

```

        Dim ea As New EndpointAddress("net.tcp://machinename:8036/endpoint")
        Dim cc As New CalculatorClient(b, ea)
        cc.Open()

```

```

        ' Alternatively, use a binding name from a configuration file generated by the
        ' SvcUtil.exe tool to create the client. Omit the binding and endpoint address
        ' because that information is provided by the configuration file.
        ' CalculatorClass cc = new CalculatorClient("ICalculator_Binding")

```

```

    End Sub

```

```

End Class 'Program

```

```

<System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0"),

```

```

System.ServiceModel.ServiceContractAttribute([Namespace] := "http://Microsoft.ServiceModel.Samples",
ConfigurationName := "ICalculator")> _
Public Interface ICalculator

    <System.ServiceModel.OperationContractAttribute(Action :=
"http://Microsoft.ServiceModel.Samples/ICalculator/Add", ReplyAction :=
"http://Microsoft.ServiceModel.Samples/ICalculator/AddResponse")> _
        Function Add(ByVal n1 As Double, ByVal n2 As Double) As Double

    <System.ServiceModel.OperationContractAttribute(Action :=
"http://Microsoft.ServiceModel.Samples/ICalculator/Subtract", ReplyAction :=
"http://Microsoft.ServiceModel.Samples/ICalculator/SubtractResponse")> _
        Function Subtract(ByVal n1 As Double, ByVal n2 As Double) As Double

    <System.ServiceModel.OperationContractAttribute(Action :=
"http://Microsoft.ServiceModel.Samples/ICalculator/Multiply", ReplyAction :=
"http://Microsoft.ServiceModel.Samples/ICalculator/MultiplyResponse")> _
        Function Multiply(ByVal n1 As Double, ByVal n2 As Double) As Double

    <System.ServiceModel.OperationContractAttribute(Action :=
"http://Microsoft.ServiceModel.Samples/ICalculator/Divide", ReplyAction :=
"http://Microsoft.ServiceModel.Samples/ICalculator/DivideResponse")> _
        Function Divide(ByVal n1 As Double, ByVal n2 As Double) As Double
End Interface 'ICalculator

<System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")> _
Public Interface ICalculatorChannel
    : Inherits ICalculator, System.ServiceModel.IClientChannel
End Interface 'ICalculatorChannel

<System.Diagnostics.DebuggerStepThroughAttribute(),
System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")> _
Public Class CalculatorClient
    Inherits System.ServiceModel.ClientBase(Of ICalculator)
    Implements ICalculator

    Public Sub New()
        ,
    End Sub 'New

    Public Sub New(ByVal endpointConfigurationName As String)
        MyBase.New(endpointConfigurationName)
    End Sub 'New

    Public Sub New(ByVal endpointConfigurationName As String, ByVal remoteAddress As String)
        MyBase.New(endpointConfigurationName, remoteAddress)
    End Sub 'New

    Public Sub New(ByVal endpointConfigurationName As String, ByVal remoteAddress As
System.ServiceModel.EndpointAddress)
        MyBase.New(endpointConfigurationName, remoteAddress)
    End Sub 'New

    Public Sub New(ByVal binding As System.ServiceModel.Channels.Binding, ByVal remoteAddress As
System.ServiceModel.EndpointAddress)
        MyBase.New(binding, remoteAddress)
    End Sub 'New

    Public Function Add(ByVal n1 As Double, ByVal n2 As Double) As Double Implements ICalculator.Add
        Return MyBase.Channel.Add(n1, n2)

```

```
End Function 'Add

Public Function Subtract(ByVal n1 As Double, ByVal n2 As Double) As Double Implements ICalculator.Subtract
    Return MyBase.Channel.Subtract(n1, n2)

End Function 'Subtract

Public Function Multiply(ByVal n1 As Double, ByVal n2 As Double) As Double Implements ICalculator.Multiply
    Return MyBase.Channel.Multiply(n1, n2)

End Function 'Multiply

Public Function Divide(ByVal n1 As Double, ByVal n2 As Double) As Double Implements ICalculator.Divide
    Return MyBase.Channel.Divide(n1, n2)

End Function 'Divide
End Class 'CalculatorClient
```

See Also

[WSHttpBinding](#)

[ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#)

[How to: Create a Client](#)

[Securing Services](#)

[Security Overview](#)

How to: Set the Security Mode

10/24/2018 • 3 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) security has three common security modes that are found on most predefined bindings: transport, message, and "transport with message credential." Two additional modes are specific to two bindings: the "transport-credential only" mode found on the [BasicHttpBinding](#), and the "Both" mode, found on the [NetMsmqBinding](#). However, this topic concentrates on the three common security modes: [Transport](#), [Message](#), and [TransportWithMessageCredential](#).

Note that not every predefined binding supports all of these modes. This topic sets the mode with the [WSHttpBinding](#) and [NetTcpBinding](#) classes and demonstrates how to set the mode both programmatically and through configuration.

For more information, see WCF security, see [Security Overview](#), [Securing Services](#), and [Securing Services and Clients](#). For more information about transport mode and message, see [Transport Security](#) and [Message Security](#).

To set the security mode in code

1. Create an instance of the binding class that you are using. For a list of predefined bindings, see [System-Provided Bindings](#). This example creates an instance of the [WSHttpBinding](#) class.
2. Set the `Mode` property of the object returned by the `Security` property.

```
WSHttpBinding b = new WSHttpBinding();  
b.Security.Mode = SecurityMode.Transport;
```

```
Dim b As New WSHttpBinding()  
b.Security.Mode = SecurityMode.Transport
```

Alternatively, set the mode to message, as shown in the following code.

```
WSHttpBinding b = new WSHttpBinding();  
b.Security.Mode = SecurityMode.Message;
```

```
Dim b As New WSHttpBinding()  
b.Security.Mode = SecurityMode.Message
```

Or set the mode to transport with message credentials, as shown in the following code.

```
WSHttpBinding b = new WSHttpBinding();  
b.Security.Mode = SecurityMode.TransportWithMessageCredential;
```

```
Dim b As New WSHttpBinding()  
b.Security.Mode = SecurityMode.TransportWithMessageCredential
```

3. You can also set the mode in the constructor of the binding, as shown in the following code.

```
WSHttpBinding b = new WSHttpBinding(SecurityMode.Message);
```

```
Dim b As New WSHttpBinding(SecurityMode.Message)
```

Setting the ClientCredentialType Property

Setting the mode to one of the three values determines how you set the `ClientCredentialType` property. For example, using the [WSHttpBinding](#) class, setting the mode to `Transport` means you must set the `ClientCredentialType` property of the [HttpTransportSecurity](#) class to an appropriate value.

To set the ClientCredentialType property for Transport mode

1. Create an instance of the binding.
2. Set the `Mode` property to `Transport`.
3. Set the `ClientCredential` property to an appropriate value. The following code sets the property to `Windows`.

```
WSHttpBinding b = new WSHttpBinding();  
b.Security.Mode = SecurityMode.Transport;  
b.Security.Transport.ClientCredentialType = HttpClientCredentialType.Windows;
```

```
Dim b As New WSHttpBinding()  
b.Security.Mode = SecurityMode.Transport  
b.Security.Transport.ClientCredentialType = HttpClientCredentialType.Windows
```

To set the ClientCredentialType property for Message mode

1. Create an instance of the binding.
2. Set the `Mode` property to `Message`.
3. Set the `ClientCredential` property to an appropriate value. The following code sets the property to `Certificate`.

```
WSHttpBinding b = new WSHttpBinding();  
b.Security.Mode = SecurityMode.Message;  
b.Security.Message.ClientCredentialType = MessageCredentialType.Certificate;
```

```
Dim b As New WSHttpBinding()  
b.Security.Mode = SecurityMode.Message  
b.Security.Message.ClientCredentialType = MessageCredentialType.Certificate
```

To set the Mode and ClientCredentialType property in configuration

1. Add an appropriate binding element to the `<bindings>` element of the configuration file. The following example adds a `<wsHttpBinding>` element.
2. Add a `<binding>` element and set its `name` attribute to an appropriate value.
3. Add a `<security>` element and set the `mode` attribute to `Message`, `Transport`, or `TransportWithMessageCredential`.
4. If the mode is set to `Transport`, add a `<transport>` element and set the `clientCredential` attribute to an appropriate value.

The following example sets the mode to `"Transport"`, and then sets the `clientCredential` attribute of the `<transport>` element to `"Windows"`.

```

<wsHttpBinding>
  <binding name="TransportSecurity">
    <security mode="Transport" >
      <transport clientCredentialType = "Windows" />
    </security>
  </binding>
</wsHttpBinding >

```

Alternatively, set the `security mode` to `"Message"`, followed by a `<"message">` element. This example sets the `clientCredentialType` to `"Certificate"`.

```

<wsHttpBinding>
  <binding name="MessageSecurity">
    <security mode="Message" >
      <message clientCredentialType = "Certificate" />
    </security>
  </binding>
</wsHttpBinding >

```

Using the [TransportWithMessageCredential](#) value is a special case, and is explained below.

Using TransportWithMessageCredential

When setting the security mode to `TransportWithMessageCredential`, the transport determines the actual mechanism that provides the transport-level security. For example, the HTTP protocol uses Secure Sockets Layer (SSL) over HTTP (HTTPS). Therefore, setting the `ClientCredentialType` property of any transport security object (such as [HttpTransportSecurity](#)) is ignored. In other words, you can only set the `ClientCredentialType` of the message security object (for the `WSHttpBinding` binding, the [NonDualMessageSecurityOverHttp](#) object).

For more information, see [How to: Use Transport Security and Message Credentials](#).

See Also

[How to: Configure a Port with an SSL Certificate](#)

[How to: Use Transport Security and Message Credentials](#)

[Transport Security](#)

[Message Security](#)

[Security Overview](#)

[System-Provided Bindings](#)

[<security>](#)

[<security>](#)

[<security>](#)

How to: Specify the Client Credential Type

5/4/2018 • 2 minutes to read • [Edit Online](#)

After setting a security mode (either transport or message), you have the option of setting the client credential type. This property specifies what type of credential the client must provide to the service for authentication. For more information about setting the security mode (a necessary step before setting the client credential type), see [How to: Set the Security Mode](#).

To set the client credential type in code

1. Create an instance of the binding that the service will use. This example uses the [WSHttpBinding](#) binding.
2. Set the [Mode](#) property to an appropriate value. This example uses the Message mode.
3. Set the [ClientCredentialType](#) property to an appropriate value. This example sets it to use Windows authentication ([Windows](#)).

```
ServiceHost myServiceHost = new ServiceHost(typeof(CalculatorService));  
// Create a binding to use.  
WSHttpBinding binding = new WSHttpBinding();  
binding.Security.Mode = SecurityMode.Message;  
binding.Security.Message.ClientCredentialType =  
    MessageCredentialType.Windows;
```

```
Dim myServiceHost As New ServiceHost(GetType(CalculatorService))  
' Create a binding to use.  
Dim binding As New WSHttpBinding()  
binding.Security.Mode = SecurityMode.Message  
binding.Security.Message.ClientCredentialType = _  
    MessageCredentialType.Windows
```

To set the client credential type in configuration

1. Add a `<system.serviceModel>` element to the configuration file.
2. As a child element, add a `<bindings>` element.
3. Add an appropriate binding. This example uses the `<wsHttpBinding>` element.
4. Add a `<binding>` element and set the `name` attribute to an appropriate value. This example uses the name "SecureBinding".
5. Add a `<security>` binding. Set the `mode` attribute to an appropriate value. This example sets it to "Message".
6. Add either a `<message>` or `<transport>` element, as determined by the security mode. Set the `clientCredentialType` attribute to an appropriate value. This example uses "Windows".

```
<system.serviceModel>
  <bindings>
    <wsHttpBinding>
      <binding name="SecureBinding">
        <security mode="Message">
          <message clientCredentialType="Windows" />
        </security>
      </binding>
    </wsHttpBinding>
  </bindings>
</system.serviceModel>
```

See Also

[Securing Services](#)

[How to: Set the Security Mode](#)

How to: Restrict Access with the PrincipalPermissionAttribute Class

5/5/2018 • 3 minutes to read • [Edit Online](#)

Controlling the access to resources on a Windows-domain computer is a basic security task. For example, only certain users should be able to view sensitive data, such as payroll information. This topic explains how to restrict access to a method by demanding that the user belong to a predefined group. For a working sample, see [Authorizing Access to Service Operations](#).

The task consists of two separate procedures. The first creates the group and populates it with users. The second applies the [PrincipalPermissionAttribute](#) class to specify the group.

To create a Windows group

1. Open the **Computer Management** console.
2. In the left panel, click **Local Users and Groups**.
3. Right-click **Groups**, and click **New Group**.
4. In the **Group Name** box, type a name for the new group.
5. In the **Description** box, type a description of the new group.
6. Click the **Add** button to add new members to the group.
7. If you have added yourself to the group and want to test the following code, you must log off the computer and log back on to be included in the group.

To demand user membership

1. Open the Windows Communication Foundation (WCF) code file that contains the implemented service contract code. For more information about implementing a contract, see [Implementing Service Contracts](#).
2. Apply the [PrincipalPermissionAttribute](#) attribute to each method that must be restricted to a specific group. Set the [Action](#) property to [Demand](#) and the [Role](#) property to the name of the group. For example:

```
// Only members of the CalculatorClients group can call this method.
[PrincipalPermission(SecurityAction.Demand, Role = "CalculatorClients")]
public double Add(double a, double b)
{
    return a + b;
}
```

```
' Only members of the CalculatorClients group can call this method.
<PrincipalPermission(SecurityAction.Demand, Role := "CalculatorClients")> _
Public Function Add(ByVal a As Double, ByVal b As Double) As Double
    Return a + b
End Function
```

NOTE

If you apply the [PrincipalPermissionAttribute](#) attribute to a contract a [SecurityException](#) will be thrown. You can only apply the attribute at the method level.

Using a Certificate to Control Access to a Method

You can also use the `PrincipalPermissionAttribute` class to control access to a method if the client credential type is a certificate. To do this, you must have the certificate's subject and thumbprint.

To examine a certificate for its properties, see [How to: View Certificates with the MMC Snap-in](#). To find the thumbprint value, see [How to: Retrieve the Thumbprint of a Certificate](#).

To control access using a certificate

1. Apply the [PrincipalPermissionAttribute](#) class to the method you want to restrict access to.
2. Set the action of the attribute to [SecurityAction.Demand](#).
3. Set the `Name` property to a string that consists of the subject name and the certificate's thumbprint. Separate the two values with a semicolon and a space, as shown in the following example:

```
// Only a client authenticated with a valid certificate that has the
// specified subject name and thumbprint can call this method.
[PrincipalPermission(SecurityAction.Demand,
    Name = "CN=ReplaceWithSubjectName; 123456712345677E8E230FDE624F841B1CE9D41E")]
public double Multiply(double a, double b)
{
    return a * b;
}
```

```
' Only a client authenticated with a valid certificate that has the
' specified subject name and thumbprint can call this method.
<PrincipalPermission(SecurityAction.Demand, Name := "CN=ReplaceWithSubjectName;
123456712345677E8E230FDE624F841B1CE9D41E")> _
Public Function Multiply(ByVal a As Double, ByVal b As Double) As Double
    Return a * b
End Function
```

4. Set the [PrincipalPermissionMode](#) property to [UseAspNetRoles](#) as shown in the following configuration example:

```
<behaviors>
  <serviceBehaviors>
    <behavior name="SvcBehavior1">
      <serviceAuthorization principalPermissionMode="UseAspNetRoles" />
    </behavior>
  </serviceBehaviors>
</behaviors>
```

Setting this value to `UseAspNetRoles` indicates that the `Name` property of the `PrincipalPermissionAttribute` will be used to perform a string comparison. When a certificate is used as a client credential, by default WCF concatenates the certificate common name and the thumbprint with a semicolon to create a unique value for the client's primary identity. With `UseAspNetRoles` set as the `PrincipalPermissionMode` on the service, this primary identity value is compared with the `Name` property value to determine the access rights of the user.

Alternatively, when creating a self-hosted service, set the [PrincipalPermissionMode](#) property in code as shown in the following code:

```
ServiceHost myServiceHost = new ServiceHost(typeof(Calculator), baseUri);
ServiceAuthorizationBehavior myServiceBehavior =
    myServiceHost.Description.Behaviors.Find<ServiceAuthorizationBehavior>();
myServiceBehavior.PrincipalPermissionMode =
    PrincipalPermissionMode.UseAspNetRoles;
```

```
Dim myServiceBehavior As ServiceAuthorizationBehavior
myServiceBehavior = _
    myServiceHost.Description.Behaviors.Find(Of ServiceAuthorizationBehavior)()
myServiceBehavior.PrincipalPermissionMode = _
    PrincipalPermissionMode.UseAspNetRoles
```

See Also

[PrincipalPermissionAttribute](#)

[PrincipalPermissionAttribute](#)

[Demand](#)

[Role](#)

[Authorizing Access to Service Operations](#)

[Security Overview](#)

[Implementing Service Contracts](#)

How to: Impersonate a Client on a Service

5/5/2018 • 2 minutes to read • [Edit Online](#)

Impersonating a client on a Windows Communication Foundation (WCF) service enables the service to perform actions on behalf of the client. For actions subject to access control list (ACL) checks, such as access to directories and files on a machine or access to a SQL Server database, the ACL check is against the client user account. This topic shows the basic steps required to enable a client in a Windows domain to set a client impersonation level. For a working example of this, see [Impersonating the Client](#). For more information about client impersonation, see [Delegation and Impersonation](#).

NOTE

When the client and service are running on the same computer and the client is running under a system account (that is, `Local System` or `Network Service`), the client cannot be impersonated when a secure session is established with stateful Security Context tokens. A WinForms or console application typically is run under the currently logged in account, so that account can be impersonated by default. However, when the client is an ASP.NET page and that page is hosted in IIS 6.0 or IIS 7.0, then the client does run under the `Network Service` account by default. All of the system-provided bindings that support secure sessions use a stateless Security Context token by default. However, if the client is an ASP.NET page and secure sessions with stateful Security Context tokens are used, the client cannot be impersonated. For more information about using stateful Security Context tokens in a secure session, see [How to: Create a Security Context Token for a Secure Session](#).

To enable impersonation of a client from a cached Windows token on a service

1. Create the service. For a tutorial of this basic procedure, see [Getting Started Tutorial](#).
2. Use a binding that uses Windows authentication and creates a session, such as [NetTcpBinding](#) or [WSHttpBinding](#).
3. When creating the implementation of the service's interface, apply the [OperationBehaviorAttribute](#) class to the method that requires client impersonation. Set the [Impersonation](#) property to [Required](#).

```
[OperationBehavior(Impersonation=ImpersonationOption.Required)]
public double Add(double a, double b)
{
    return a + b;
}
```

```
<OperationBehavior(Impersonation := ImpersonationOption.Required)> _
Public Function Add(ByVal a As Double, ByVal b As Double) As Double _
    Implements ICalculator.Add
    Return a + b
End Function
```

To set the allowed impersonation level on the client

1. Create service client code by using the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#). For more information, see [Accessing Services Using a WCF Client](#).
2. After creating the WCF client, set the [AllowedImpersonationLevel](#) property of the [WindowsClientCredential](#) class to one of the [TokenImpersonationLevel](#) enumeration values.

NOTE

To use [Delegation](#), negotiated Kerberos authentication (sometimes called *multi-leg* or *multi-step* Kerberos) must be used. For a description of how to implement this, see [Best Practices for Security](#).

```
CalculatorClient client = new CalculatorClient("CalculatorEndpoint");  
client.ClientCredentials.Windows.AllowedImpersonationLevel =  
    System.Security.Principal.TokenImpersonationLevel.Impersonation;
```

```
Dim client As New CalculatorClient("CalculatorEndpoint")  
client.ClientCredentials.Windows.AllowedImpersonationLevel = _  
    System.Security.Principal.TokenImpersonationLevel.Impersonation
```

See Also

[OperationBehaviorAttribute](#)

[TokenImpersonationLevel](#)

[Impersonating the Client](#)

[Delegation and Impersonation](#)

How to: Examine the Security Context

10/24/2018 • 2 minutes to read • [Edit Online](#)

When programming Windows Communication Foundation (WCF) services, the service security context enables you to determine details about the client credentials and claims used to authenticate with the service. This is done by using the properties of the [ServiceSecurityContext](#) class.

For example, you can retrieve the identity of the current client by using the [PrimaryIdentity](#) or the [WindowsIdentity](#) property. To determine whether the client is anonymous, use the [IsAnonymous](#) property.

You can also determine what claims are being made on behalf of the client by iterating through the collection of claims in the [AuthorizationContext](#) property.

To get the current security context

- Access the static property [Current](#) to get the current security context. Examine any of the properties of the current context from the reference.

To determine the identity of the caller

1. Print the value of the [PrimaryIdentity](#) and [WindowsIdentity](#) properties.

To parse the claims of a caller

1. Return the current [AuthorizationContext](#) class. Use the [Current](#) property to return the current service security context, then return the `AuthorizationContext` using the [AuthorizationContext](#) property.
2. Parse the collection of [ClaimSet](#) objects returned by the [ClaimSets](#) property of the [AuthorizationContext](#) class.

Example

The following example prints the values of the [WindowsIdentity](#) and [PrimaryIdentity](#) properties of the current security context and the [ClaimType](#) property, the resource value of the claim, and the [Right](#) property of every claim in the current security context.

```
// Run this method from within a method protected by the PrincipalPermissionAttribute
// to see the security context data, including the primary identity.
public void WriteServiceSecurityContextData(string fileName)
{
    using (StreamWriter sw = new StreamWriter(fileName))
    {
        // Write the primary identity and Windows identity. The primary identity is derived from
        // the credentials used to authenticate the user. The Windows identity may be a null string.
        sw.WriteLine("PrimaryIdentity: {0}", ServiceSecurityContext.Current.PrimaryIdentity.Name);
        sw.WriteLine("WindowsIdentity: {0}", ServiceSecurityContext.Current.WindowsIdentity.Name);
        sw.WriteLine();
        // Write the claimsets in the authorization context. By default, there is only one claimset
        // provided by the system.
        foreach (ClaimSet claimset in ServiceSecurityContext.Current.AuthorizationContext.ClaimSets)
        {
            foreach (Claim claim in claimset)
            {
                // Write out each claim type, claim value, and the right. There are two
                // possible values for the right: "identity" and "possessproperty".
                sw.WriteLine("Claim Type = {0}", claim.ClaimType);
                sw.WriteLine("\t Resource = {0}", claim.Resource.ToString());
                sw.WriteLine("\t Right = {0}", claim.Right);
            }
        }
    }
}
```

```
' Run this method from within a method protected by the PrincipalPermissionAttribute
' to see the security context data, including the primary identity.
Public Sub WriteServiceSecurityContextData(ByVal fileName As String)
    Dim sw As New StreamWriter(fileName)
    Try
        ' Write the primary identity and Windows identity. The primary identity is derived from
        ' the credentials used to authenticate the user. The Windows identity may be a null string.
        sw.WriteLine("PrimaryIdentity: {0}", ServiceSecurityContext.Current.PrimaryIdentity.Name)
        sw.WriteLine("WindowsIdentity: {0}", ServiceSecurityContext.Current.WindowsIdentity.Name)
        sw.WriteLine()
        ' Write the claimsets in the authorization context. By default, there is only one claimset
        ' provided by the system.
        Dim claimset As ClaimSet
        For Each claimset In ServiceSecurityContext.Current.AuthorizationContext.ClaimSets
            Dim claim As Claim
            For Each claim In claimset
                ' Write out each claim type, claim value, and the right. There are two
                ' possible values for the right: "identity" and "possessproperty".
                sw.WriteLine("Claim Type = {0}", claim.ClaimType)
                sw.WriteLine(vbTab + " Resource = {0}", claim.Resource.ToString())
                sw.WriteLine(vbTab + " Right = {0}", claim.Right)
            Next claim
        Next claimset
    Finally
        sw.Dispose()
    End Try
End Sub
```

Compiling the Code

The code uses the following namespaces:

- [System](#)

- [System.ServiceModel](#)
- [System.IdentityModel.Policy](#)
- [System.IdentityModel.Claims](#)

See Also

[Securing Services](#)

[Service Identity and Authentication](#)

Understanding Protection Level

5/5/2018 • 6 minutes to read • [Edit Online](#)

The `ProtectionLevel` property is found on many different classes, such as the [ServiceContractAttribute](#) and the [OperationContractAttribute](#) classes. The property controls how a part (or whole) of a message is protected. This topic explains the Windows Communication Foundation (WCF) feature and how it works.

For instructions on setting the protection level, see [How to: Set the ProtectionLevel Property](#).

NOTE

Protection levels can be set only in code, not in configuration.

Basics

To understand the protection level feature, the following basic statements apply:

- Three basic levels of protection exist for any part of a message. The property (wherever it occurs) is set to one of the [ProtectionLevel](#) enumeration values. In ascending order of protection, they include:
 - `None` .
 - `Sign` . The protected part is digitally signed. This ensures detection of any tampering with the protected message part.
 - `EncryptAndSign` . The message part is encrypted to ensure confidentiality before it is signed.
- You can set protection requirements only for *application data* with this feature. For example, WS-Addressing headers are infrastructure data and, therefore, are not affected by the `ProtectionLevel` .
- When the security mode is set to `Transport` , the entire message is protected by the transport mechanism. Therefore, setting a separate protection level for different parts of a message has no effect.
- The `ProtectionLevel` is a way for the developer to set the *minimum level* that a binding must comply with. When a service is deployed, the actual binding specified in configuration may or may not support the minimum level. For example, by default, the [BasicHttpBinding](#) class does not supply security (although it can be enabled). Therefore, using it with a contract that has any setting other than `None` will cause an exception to be thrown.
- If the service requires that the minimum `ProtectionLevel` for all messages is `Sign` , a client (perhaps created by a non-WCF technology) can encrypt and sign all messages (which is more than the minimum required). In this case, WCF will not throw an exception because the client has done more than the minimum. Note, however, that WCF applications (services or clients) will not over-secure a message part if possible but will comply with the minimum level. Also note that when using `Transport` as the security mode, the transport may over-secure the message stream because it is inherently unable to secure at a more granular level.
- If you set the `ProtectionLevel` explicitly to either `Sign` or `EncryptAndSign` , then you must use a binding with security enabled or an exception will be thrown.
- If you select a binding that enables security and you do not set the `ProtectionLevel` property anywhere on the contract, all application data will be encrypted and signed.

- If you select a binding that does not have security enabled (for example, the `BasicHttpBinding` class has security disabled by default), and the `ProtectionLevel` is not explicitly set, then none of the application data will be protected.
- If you are using a binding that applies security at the transport level, all application data will be secured according to the capabilities of the transport.
- If you use a binding that applies security at the message level, then application data will be secured according to the protection levels set on the contract. If you do not specify a protection level, then all application data in the messages will be encrypted and signed.
- The `ProtectionLevel` can be set at different scoping levels. There is a hierarchy associated with scoping, which is explained in the next section.

Scoping

Setting the `ProtectionLevel` on the topmost API sets the level for all levels below it. If the `ProtectionLevel` is set to a different value at a lower level, all APIs below that level in the hierarchy will now be reset to the new level (APIs above it, however, will still be affected by the topmost level). The hierarchy is as follows. Attributes at the same level are peers.

[ServiceContractAttribute](#)

[OperationContractAttribute](#)

[FaultContractAttribute](#)

[MessageContractAttribute](#)

[MessageHeaderAttribute](#)

[MessageBodyMemberAttribute](#)

Programming ProtectionLevel

To program the `ProtectionLevel` at any point in the hierarchy, simply set the property to an appropriate value when applying the attribute. For examples, see [How to: Set the ProtectionLevel Property](#).

NOTE

Setting the property on faults and message contracts requires understanding how those features work. For more information, see [How to: Set the ProtectionLevel Property](#) and [Using Message Contracts](#).

WS-Addressing Dependency

In most cases, using the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) to generate a client ensures that the client and service contracts are identical. However, seemingly identical contracts can cause the client to throw an exception. This occurs whenever a binding does not support the WS-Addressing specification and multiple levels of protection are specified on the contract. For example, the `BasicHttpBinding` class does not support the specification, or if you create a custom binding that does not support WS-Addressing. The `ProtectionLevel` feature relies on the WS-Addressing specification to enable different protection levels on a single contract. If the binding does not support the WS-Addressing specification, all levels will be set to the same protection level. The effective protection level for all scopes on the contract will be set to the strongest protection level used on the contract.

This may cause a problem that is hard to debug at first glance. It is possible to create a client contract (an

interface) that includes methods for more than one service. That is, the same interface is used to create a client that communicates with many services, and the single interface contains methods for all services. The developer must take care in this rare scenario to invoke only those methods that are applicable for each particular service. If the binding is the [BasicHttpBinding](#) class, multiple protection levels cannot be supported. However, a service replying to the client might respond to a client with a lower protection level than required. In this case, the client will throw an exception because it expects a higher protection level.

An example of the code illustrates this problem. The following example shows a service and a client contract. Assume that the binding is the [<basicHttpBinding>](#) element. Therefore, all operations on a contract have the same protection level. This uniform protection level is determined as the maximum protection level across all operations.

The service contract is:

```
[ServiceContract()]
public interface IPurchaseOrder
{
    [OperationContract(ProtectionLevel = ProtectionLevel.Sign)]
    int Price();
}
```

```
<ServiceContract> _
Public Interface IPurchaseOrder
    <OperationContract(ProtectionLevel := ProtectionLevel.Sign)> _
    Function Price() As Integer
End Interface
```

The following code shows the client contract interface. Note that it includes a `Tax` method that is intended to be used with a different service:

```
[ServiceContract()]
public interface IPurchaseOrder
{
    [OperationContract()]
    int Tax();

    [OperationContract(ProtectionLevel = ProtectionLevel.Sign)]
    int Price();
}
```

```
<ServiceContract> _
Public Interface IPurchaseOrder
    <OperationContract> _
    Function Tax() As Integer

    <OperationContract(ProtectionLevel := ProtectionLevel.Sign)> _
    Function Price() As Integer
End Interface
```

When the client calls the `Price` method, it throws an exception when it receives a reply from the service. This occurs because the client does not specify a `ProtectionLevel` on the `ServiceContractAttribute`, and therefore the client uses the default ([EncryptAndSign](#)) for all methods, including the `Price` method. However, the service returns the value using the [Sign](#) level because the service contract defines a single method that has its protection level set to [Sign](#). In this case, the client will throw an error when validating the response from the service.

See Also

[ServiceContractAttribute](#)

[OperationContractAttribute](#)

[FaultContractAttribute](#)

[MessageContractAttribute](#)

[MessageHeaderAttribute](#)

[MessageBodyMemberAttribute](#)

[ProtectionLevel](#)

[Securing Services](#)

[How to: Set the ProtectionLevel Property](#)

[Specifying and Handling Faults in Contracts and Services](#)

[Using Message Contracts](#)

How to: Set the ProtectionLevel Property

5/4/2018 • 6 minutes to read • [Edit Online](#)

You can set the protection level by applying an appropriate attribute and setting the property. You can set protection at the service level to affect all parts of every message, or you can set protection at increasingly granular levels, from methods to message parts. For more information about the `ProtectionLevel` property, see [Understanding Protection Level](#).

NOTE

You can set protection levels only in code, not in configuration.

To sign all messages for a service

1. Create an interface for the service.
2. Apply the `ServiceContractAttribute` attribute to the service and set the `ProtectionLevel` property to `Sign`, as shown in the following code (the default level is `EncryptAndSign`).

```
// Set the ProtectionLevel on the whole service to Sign.
[ServiceContract(ProtectionLevel = ProtectionLevel.Sign)]
public interface ICalculator
```

```
' Set the ProtectionLevel on the whole service to Sign.
<ServiceContract(ProtectionLevel:=ProtectionLevel.Sign)> _
Public Interface ICalculator
```

To sign all message parts for an operation

1. Create an interface for the service and apply the `ServiceContractAttribute` attribute to the interface.
2. Add a method declaration to the interface.
3. Apply the `OperationContractAttribute` attribute to the method, and set the `ProtectionLevel` property to `Sign`, as shown in the following code.

```
// Set the ProtectionLevel on the whole service to Sign.
[ServiceContract(ProtectionLevel = ProtectionLevel.Sign)]
public interface ICalculator
{
    // Set the ProtectionLevel on this operation to None.
    [OperationContract(ProtectionLevel = ProtectionLevel.Sign)]
    double Add(double a, double b);
}
```

```

' Set the ProtectionLevel on the whole service to Sign.
<ServiceContract(ProtectionLevel:=ProtectionLevel.Sign)> _
Public Interface ICalculator

    ' Set the ProtectionLevel on this operation to Sign.
    <OperationContract(ProtectionLevel:=ProtectionLevel.Sign)> _
    Function Add(ByVal a As Double, ByVal b As Double) As Double
End Interface

```

Protecting Fault Messages

Exceptions that are thrown on a service can be sent to a client as SOAP faults. For more information about creating strongly typed faults, see [Specifying and Handling Faults in Contracts and Services](#) and [How to: Declare Faults in Service Contracts](#).

To protect a fault message

1. Create a type that represents the fault message. The following example creates a class named `MathFault` with two fields.
2. Apply the [DataContractAttribute](#) attribute to the type and a [DataMemberAttribute](#) attribute to each field that should be serialized, as shown in the following code.

```

[DataContract]
public class MathFault
{
    [DataMember]
    public string operation;
    [DataMember]
    public string description;
}

```

```

<DataContract()> _
Public Class MathFault
    <DataMember()> _
    Public operation As String
    <DataMember()> _
    Public description As String
End Class

```

3. In the interface that will return the fault, apply the [FaultContractAttribute](#) attribute to the method that will return the fault and set the `detailType` parameter to the type of the fault class.
4. Also in the constructor, set the [ProtectionLevel](#) property to [EncryptAndSign](#), as shown in the following code.

```

public interface ICalculator
{
    // Set the ProtectionLevel on a FaultContractAttribute.
    [OperationContract(ProtectionLevel = ProtectionLevel.EncryptAndSign)]
    [FaultContract(
        typeof(MathFault),
        Action = @"http://localhost/Add",
        Name = "AddFault",
        Namespace = "http://contoso.com",
        ProtectionLevel = ProtectionLevel.EncryptAndSign)]
    double Add(double a, double b);
}

```

```
Public Interface ICalculator
    ' Set the ProtectionLevel on a FaultContractAttribute.
    <OperationContract(ProtectionLevel := ProtectionLevel.EncryptAndSign), _
        FaultContract(GetType(MathFault), ProtectionLevel := ProtectionLevel.EncryptAndSign)> _
    Function Add(ByVal a As Double, ByVal b As Double) As Double
End Interface
```

Protecting Message Parts

Use a message contract to protect parts of a message. For more information about message contracts, see [Using Message Contracts](#).

To protect a message body

1. Create a type that represents the message. The following example creates a `Company` class with two fields, `CompanyName` and `CompanyID`.
2. Apply the [MessageContractAttribute](#) attribute to the class and set the [ProtectionLevel](#) property to [EncryptAndSign](#).
3. Apply the [MessageHeaderAttribute](#) attribute to a field that will be expressed as a message header and set the `ProtectionLevel` property to [EncryptAndSign](#).
4. Apply the [MessageBodyMemberAttribute](#) to any field that will be expressed as part of the message body, and set the `ProtectionLevel` property to [EncryptAndSign](#), as shown in the following example.

```
[MessageContract(ProtectionLevel = ProtectionLevel.EncryptAndSign)]
public class Company
{
    [MessageHeader(ProtectionLevel = ProtectionLevel.Sign)]
    public string CompanyName;

    [MessageBodyMember(ProtectionLevel = ProtectionLevel.EncryptAndSign)]
    public string CompanyID;
}
```

```
<MessageContract(ProtectionLevel := ProtectionLevel.EncryptAndSign)> _
Public Class Company
    <MessageHeader(ProtectionLevel := ProtectionLevel.Sign)> _
    Public CompanyName As String

    <MessageBodyMember(ProtectionLevel := ProtectionLevel.EncryptAndSign)> _
    Public CompanyID As String
End Class
```

Example

The following example sets the `ProtectionLevel` property of several attribute classes at various places in a service.

```
[ServiceContract(ProtectionLevel = ProtectionLevel.EncryptAndSign)]
public interface ICalculator
{
    [OperationContract(ProtectionLevel = ProtectionLevel.Sign)]
    double Add(double a, double b);

    [OperationContract()]
    [FaultContract(typeof(MathFault),
```

```

        ProtectionLevel = ProtectionLevel.EncryptAndSign]]
        double Subtract(double a, double b);

        [OperationContract(ProtectionLevel = ProtectionLevel.EncryptAndSign)]
        Company GetCompanyInfo();
    }

    [DataContract]
    public class MathFault
    {
        [DataMember]
        public string operation;
        [DataMember]
        public string description;
    }

    [MessageContract(ProtectionLevel = ProtectionLevel.EncryptAndSign)]
    public class Company
    {
        [MessageHeader(ProtectionLevel = ProtectionLevel.Sign)]
        public string CompanyName;

        [MessageBodyMember(ProtectionLevel = ProtectionLevel.EncryptAndSign)]
        public string CompanyID;
    }

    [MessageContract(ProtectionLevel = ProtectionLevel.Sign)]
    public class Calculator : ICalculator
    {
        public double Add(double a, double b)
        {
            return a + b;
        }

        public double Subtract(double a, double b)
        {
            return a - b;
        }

        public Company GetCompanyInfo()
        {
            Company co = new Company();
            co.CompanyName = "www.cohowinery.com";
            return co;
        }
    }

    public class Test
    {
        static void Main()
        {
            Test t = new Test();
            try
            {
                t.Run();
            }
            catch (System.InvalidOperationException inv)
            {
                Console.WriteLine(inv.Message);
            }
        }

        private void Run()
        {
            // Create a binding and set the security mode to Message.
            WSHttpBinding b = new WSHttpBinding();

```

```

    b.Security.Mode = SecurityMode.Message;

    Type contractType = typeof(ICalculator);
    Type implementedContract = typeof(Calculator);
    Uri baseAddress = new Uri("http://localhost:8044/base");

    // Create the ServiceHost and add an endpoint.
    ServiceHost sh = new ServiceHost(implementedContract, baseAddress);
    sh.AddServiceEndpoint(contractType, b, "Calculator");

    ServiceMetadataBehavior sm = new ServiceMetadataBehavior();
    sm.HttpGetEnabled = true;
    sh.Description.Behaviors.Add(sm);
    sh.Credentials.ServiceCertificate.SetCertificate(
        StoreLocation.CurrentUser,
        StoreName.My,
        X509FindType.FindByIssuerName,
        "ValidCertificateIssuer");

    foreach (ServiceEndpoint se in sh.Description.Endpoints)
    {
        ContractDescription cd = se.Contract;
        Console.WriteLine("\nContractDescription: ProtectionLevel = {0}", cd.Name, cd.ProtectionLevel);
        foreach (OperationDescription od in cd.Operations)
        {
            Console.WriteLine("\nOperationDescription: Name = {0}", od.Name, od.ProtectionLevel);
            Console.WriteLine("ProtectionLevel = {1}", od.Name, od.ProtectionLevel);
            foreach (MessageDescription m in od.Messages)
            {
                Console.WriteLine("\t {0}: {1}", m.Action, m.ProtectionLevel);
                foreach (MessageHeaderDescription mh in m.Headers)
                {
                    Console.WriteLine("\t\t {0}: {1}", mh.Name, mh.ProtectionLevel);

                    foreach (MessagePropertyDescription mp in m.Properties)
                    {
                        Console.WriteLine("{0}: {1}", mp.Name, mp.ProtectionLevel);
                    }
                }
            }
        }
    }
    sh.Open();
    Console.WriteLine("Listening");
    Console.ReadLine();
    sh.Close();
}
}

```

```

<ServiceContract(ProtectionLevel := ProtectionLevel.EncryptAndSign)> _
Public Interface ICalculator
    <OperationContract(ProtectionLevel := ProtectionLevel.EncryptAndSign)> _
    Function Add(ByVal a As Double, ByVal b As Double) As Double

    <OperationContract(), _
    FaultContract _
    (GetType(MathFault), _
    Action := "http://localhost/Add", _
    Name := "AddFault", _
    Namespace := "http://contoso.com", _
    ProtectionLevel := ProtectionLevel.EncryptAndSign)> _
    Function Subtract(ByVal a As Double, ByVal b As Double) As Double

    <OperationContract(ProtectionLevel := ProtectionLevel.EncryptAndSign)> _
    Function GetCompanyInfo() As Company

```

End Interface

<DataContract()> _

Public Class MathFault

 <DataMember()> _

 Public operation As String

 <DataMember()> _

 Public description As String

End Class

<MessageContract(ProtectionLevel := ProtectionLevel.EncryptAndSign)> _

Public Class Company

 <MessageHeader(ProtectionLevel := ProtectionLevel.Sign)> _

 Public CompanyName As String

 <MessageBodyMember(ProtectionLevel := ProtectionLevel.EncryptAndSign)> _

 Public CompanyID As String

End Class

<MessageContract(ProtectionLevel := ProtectionLevel.Sign)> _

Public Class Calculator

 Implements ICalculator

 Public Function Add(ByVal a As Double, ByVal b As Double) As Double _

 Implements ICalculator.Add

 Return a + b

End Function

 Public Function Subtract(ByVal a As Double, ByVal b As Double) As Double _

 Implements ICalculator.Subtract

 Return a - b

End Function

 Public Function GetCompanyInfo() As Company Implements ICalculator.GetCompanyInfo

 Dim co As New Company()

 co.CompanyName = "www.cohowinery.com"

 Return co

End Function

End Class

Public Class Test

 Shared Sub Main()

 Dim t As New Test()

 Try

 t.Run()

 Catch inv As System.InvalidOperationException

 Console.WriteLine(inv.Message)

 End Try

 End Sub

 Private Sub Run()

 ' Create a binding and set the security mode to Message.

 Dim b As New WSHttpBinding()

 b.Security.Mode = SecurityMode.Message

 Dim contractType As Type = GetType(ICalculator)

 Dim implementedContract As Type = GetType(Calculator)

 Dim baseAddress As New Uri("http://localhost:8044/base")

 ' Create the ServiceHost and add an endpoint.

 Dim sh As New ServiceHost(implementedContract, baseAddress)

 sh.AddServiceEndpoint(contractType, b, "Calculator")

```

Dim sm As New ServiceMetadataBehavior()
sm.HttpGetEnabled = True
sh.Description.Behaviors.Add(sm)
sh.Credentials.ServiceCertificate.SetCertificate( _
    StoreLocation.CurrentUser, StoreName.My, _
    X509FindType.FindByIssuerName, "ValidCertificateIssuer")

Dim se As ServiceEndpoint
For Each se In sh.Description.Endpoints
    Dim cd As ContractDescription = se.Contract
    Console.WriteLine(vbLf + "ContractDescription: ProtectionLevel = {0}", _
        cd.Name, cd.ProtectionLevel)
    Dim od As OperationDescription
    For Each od In cd.Operations
        Console.WriteLine(vbLf + "OperationDescription: Name = {0}", od.Name, od.ProtectionLevel)
        Console.WriteLine("ProtectionLevel = {1}", od.Name, od.ProtectionLevel)
        Dim m As MessageDescription
        For Each m In od.Messages
            Console.WriteLine(vbTab + " {0}: {1}", m.Action, m.ProtectionLevel)
            Dim mh As MessageHeaderDescription
            For Each mh In m.Headers
                Console.WriteLine(vbTab + vbTab + " {0}: {1}", mh.Name, mh.ProtectionLevel)

                Dim mp As MessagePropertyDescription
                For Each mp In m.Properties
                    Console.WriteLine("{0}: {1}", mp.Name, mp.ProtectionLevel)
                Next mp
            Next mh
        Next m
    Next od
Next se
sh.Open()
Console.WriteLine("Listening")
Console.ReadLine()
sh.Close()

End Sub
End Class

```

Compiling the Code

The following code shows the namespaces required to compile the example code.

```

using System;
using System.ServiceModel;
using System.Net.Security;
using System.ServiceModel.Description;
using System.Security.Permissions;
using System.Security.Cryptography.X509Certificates;
using System.Runtime.Serialization;

```

```

Imports System
Imports System.ServiceModel
Imports System.Net.Security
Imports System.ServiceModel.Description
Imports System.Security.Permissions
Imports System.Security.Cryptography.X509Certificates
Imports System.Runtime.Serialization

```

See Also

ServiceContractAttribute
OperationContractAttribute
FaultContractAttribute
MessageContractAttribute
MessageBodyMemberAttribute
Understanding Protection Level

Creating WS-I Basic Profile 1.1 Interoperable Services

10/19/2018 • 2 minutes to read • [Edit Online](#)

To configure a WCF service endpoint to be interoperable with ASP.NET Web service clients:

- Use the [System.ServiceModel.BasicHttpBinding](#) type as the binding type for your service endpoint.
- Do not use callback and session contract features or transaction behaviors on your service endpoint

You can optionally enable support for HTTPS and transport-level client authentication on the binding.

The following features of the [BasicHttpBinding](#) class require functionality beyond WS-I Basic Profile 1.1:

- Message Transmission Optimization Mechanism (MTOM) message encoding controlled by the [BasicHttpBinding.MessageEncoding](#) property. Leave this property at its default value, which is [WSMessageEncoding.Text](#) to not use MTOM.
- Message security controlled by the [BasicHttpBinding.Security](#) value provides WS-Security support compliant with WS-I Basic Security Profile 1.0. Leave this property at its default value, which is [SecurityMode.Transport](#) to not use WS-Security.

To make the metadata for a WCF service available to ASP.NET, use the Web service client generation tools: [Web Services Description Language Tool \(WSDL.exe\)](#), [Web Services Discovery Tool \(Disco.exe\)](#), and the

[Add Web Reference](#) feature in Visual Studio; you must enable metadata publication. For more information, see [Publishing Metadata Endpoints](#).

Example

Description

The following example code demonstrates how to add a WCF endpoint that is compatible with ASP.NET Web service clients in code and, alternatively, in a configuration file.

Code

```

using System;
using System.Collections.Generic;
using System.Text;
using System.ServiceModel;
using System.ServiceModel.Description;

[ServiceContract]
public interface IEcho
{
    [OperationContract]
    string Echo(string s);
}

public class MyService : IEcho
{
    public string Echo(string s)
    {
        return s;
    }
}

class Program
{
    static void Main(string[] args)
    {
        string baseAddress = "http://localhost:8080/wcfselfhost/";
        ServiceHost host = new ServiceHost(typeof(MyService), new Uri(baseAddress));

        // Create a BasicHttpBinding instance
        BasicHttpBinding binding = new BasicHttpBinding();

        // Add a service endpoint using the created binding
        host.AddServiceEndpoint(typeof(IEcho), binding, "echo1");

        host.Open();
        Console.WriteLine("Service listening on {0} . . .", baseAddress);
        Console.ReadLine();
        host.Close();
    }
}

```

```
Imports System
Imports System.Collections.Generic
Imports System.Text
Imports System.ServiceModel
Imports System.ServiceModel.Description

<ServiceContract()> _
Public Interface IEcho

    <OperationContract()> _
    Function Echo(ByVal s As String) As String

End Interface

Public Class MyService
    Implements IEcho

    Public Function Echo(ByVal s As String) As String Implements IEcho.Echo
        Return s
    End Function

End Class

Friend Class Program

    Shared Sub Main(ByVal args() As String)
        Dim baseAddress = "http://localhost:8080/wcfselfhost/"
        Dim host As New ServiceHost(GetType(MyService), _
            New Uri(baseAddress))

        ' Add a service endpoint using the created binding
        With host
            .AddServiceEndpoint(GetType(IEcho), _
                New BasicHttpBinding(), _
                "echo1")

            .Open()
            Console.WriteLine("Service listening on {0} . . .", _
                baseAddress)

            Console.ReadLine()
            .Close()
        End With
    End Sub

End Class
```

```
<configuration>
  <system.serviceModel>
    <services>
      <service name="MyService" behaviorConfiguration="HttpGetMetadata">
        <endpoint address="echo2" contract="IEcho" binding="basicHttpBinding" />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="HttpGetMetadata">
          <serviceMetadata httpGetEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

See Also

Hosting Services

5/5/2018 • 7 minutes to read • [Edit Online](#)

To become active, a service must be hosted within a run-time environment that creates it and controls its context and lifetime. Windows Communication Foundation (WCF) services are designed to run in any Windows process that supports managed code.

WCF provides a unified programming model for building service-oriented applications. This programming model remains consistent and is independent of the run-time environment in which the service is deployed. In practice, this means that the code for your services looks much the same whatever the hosting option.

These hosting options range from running inside a console application to server environments such as a Windows service running within a worker process managed by Internet Information Services (IIS) or by Windows Process Activation Service (WAS). Developers choose the hosting environment that satisfies the service's deployment requirements. These requirements might derive from the platform on which the application is deployed, the transport on which it must send and receive messages, or on the type of process recycling and other process management required to ensure adequate availability, or on some other management or reliability requirements. The next section provides information and guidance on hosting options.

Hosting Options

Self-Hosting in a Managed Application

WCF services can be hosted in any managed application. This is the most flexible option because it requires the least infrastructure to deploy. You embed the code for the service inside the managed application code and then create and open an instance of the [ServiceHost](#) to make the service available. For more information, see [How to: Host a WCF Service in a Managed Application](#).

This option enables two common scenarios: WCF services running inside console applications and rich client applications such as those based on Windows Presentation Foundation (WPF) or Windows Forms (WinForms). Hosting a WCF service inside a console application is typically useful during the application's development phase. This makes them easy to debug, easy to get trace information from to find out what is happening inside of the application, and easy to move around by copying them to new locations. This hosting option also makes it easy for rich client applications, such as WPF and WinForms applications, to communicate with the outside world. For example, a peer-to-peer collaboration client that uses WPF for its user interface and also hosts a WCF service that allows other clients to connect to it and share information.

Managed Windows Services

This hosting option consists of registering the application domain (AppDomain) that hosts an WCF service as a managed Windows Service (formerly known as NT service) so that the process lifetime of the service is controlled by the service control manager (SCM) for Windows services. Like the self-hosting option, this type of hosting environment requires that some hosting code is written as part of the application. The service is implemented as both a Windows Service and as an WCF service by causing it to inherit from the [ServiceBase](#) class as well as from an WCF service contract interface. The [ServiceHost](#) is then created and opened within an overridden [OnStart\(String\[\]\)](#) method and closed within an overridden [OnStop\(\)](#) method. An installer class that inherits from [Installer](#) must also be implemented to allow the program to be installed as a Windows Service by the Installutil.exe tool. For more information, see [How to: Host a WCF Service in a Managed Windows Service](#). The scenario enabled by the managed Windows Service hosting option is that of a long-running WCF service hosted outside of IIS in a secure environment that is not message-activated. The lifetime of the service is controlled instead by the operating system. This hosting option is available in all versions of Windows.

Internet Information Services (IIS)

The IIS hosting option is integrated with ASP.NET and uses the features these technologies offer, such as process recycling, idle shutdown, process health monitoring, and message-based activation. On the Windows XP and Windows Server 2003 operating systems, this is the preferred solution for hosting Web service applications that must be highly available and highly scalable. IIS also offers the integrated manageability that customers expect from an enterprise-class server product. This hosting option requires that IIS be properly configured, but it does not require that any hosting code be written as part of the application. For more information about how to configure IIS hosting for a WCF service, see [How to: Host a WCF Service in IIS](#).

Note that IIS-hosted services can only use the HTTP transport. Its implementation in IIS 5.1 has introduced some limitations in Windows XP. The message-based activation provided for an WCF service by IIS 5.1 on Windows XP blocks any other self-hosted WCF service on the same computer from using port 80 to communicate. WCF services can run in the same AppDomain/Application Pool/Worker Process as other applications when hosted by IIS 6.0 on Windows Server 2003. But because WCF and IIS 6.0 both use the kernel-mode HTTP stack (HTTP.sys), IIS 6.0 can share port 80 with other self-hosted WCF services running on the same machine, unlike IIS 5.1.

Windows Process Activation Service (WAS)

Windows Process Activation Service (WAS) is the new process activation mechanism for the Windows Server 2008 that is also available on Windows Vista. It retains the familiar IIS 6.0 process model (application pools and message-based process activation) and hosting features (such as rapid failure protection, health monitoring, and recycling), but it removes the dependency on HTTP from the activation architecture. IIS 7.0 uses WAS to accomplish message-based activation over HTTP. Additional WCF components also plug into WAS to provide message-based activation over the other protocols that WCF supports, such as TCP, MSMQ, and named pipes. This allows applications that use communication protocols to use the IIS features such as process recycling, rapid fail protection, and the common configuration system that were only available to HTTP-based applications.

This hosting option requires that WAS be properly configured, but it does not require you to write any hosting code as part of the application. For more information about how to configure WAS hosting, see [How to: Host a WCF Service in WAS](#).

Choosing a Hosting Environment

The following table summarizes some of the key benefits and scenarios associated with each of the hosting options.

HOSTING ENVIRONMENT	COMMON SCENARIOS	KEY BENEFITS AND LIMITATIONS
Managed Application ("Self-Hosted")	<ul style="list-style-type: none">- Console applications used during development.- Rich WinForm and WPF client applications accessing services.	<ul style="list-style-type: none">- Flexible.- Easy to deploy.- Not an enterprise solution for services.
Windows Services (formerly known as NT services)	<ul style="list-style-type: none">- A long-running WCF service hosted outside of IIS.	<ul style="list-style-type: none">- Service process lifetime controlled by the operating system, not message-activated.- Supported by all versions of Windows.- Secure environment.
IIS 5.1, IIS 6.0	<ul style="list-style-type: none">- Running a WCF service side-by-side with ASP.NET content on the Internet using the HTTP protocol.	<ul style="list-style-type: none">- Process recycling.- Idle shutdown.- Process health monitoring.- Message-based activation.- HTTP only.

HOSTING ENVIRONMENT	COMMON SCENARIOS	KEY BENEFITS AND LIMITATIONS
Windows Process Activation Service (WAS)	<ul style="list-style-type: none"> - Running a WCF service without installing IIS on the Internet using various transport protocols. 	<ul style="list-style-type: none"> - IIS is not required. - Process recycling. - Idle shutdown. - Process health monitoring. - Message-based activation. - Works with HTTP, TCP, named pipes, and MSMQ.
IIS 7.0	<ul style="list-style-type: none"> - Running a WCF service with ASP.NET content. - Running a WCF service on the Internet using various transport protocols. 	<ul style="list-style-type: none"> - WAS benefits. - Integrated with ASP.NET and IIS content.

The choice of a hosting environment depends on the version of Windows on which it is deployed, the transports it requires to send messages and the type of process and application domain recycling it requires. The following table summarizes the data related to these requirements.

HOSTING ENVIRONMENT	PLATFORM AVAILABILITY	TRANSPORTS SUPPORTED	PROCESS AND APPDOMAIN RECYCLING
Managed Applications ("Self-Hosted")	Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008	HTTP, net.tcp, net.pipe, net.msmq	No
Windows Services (formerly known as NT services)	Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008	HTTP, net.tcp, net.pipe, net.msmq	No
IIS 5.1	Windows XP	HTTP	Yes
IIS 6.0	Windows Server 2003	HTTP	Yes
Windows Process Activation Service (WAS)	Windows Vista, Windows Server 2008	HTTP, net.tcp, net.pipe, net.msmq	Yes

It is important to note that running a service or any extension from an untrusted host compromises security. Also, note that when opening a [ServiceHost](#) under impersonation, an application must ensure that the user is not logged off, for example by caching the [WindowsIdentity](#) of the user.

See Also

[System Requirements](#)

[Basic Programming Lifecycle](#)

[Implementing Service Contracts](#)

[How to: Host a WCF Service in IIS](#)

[How to: Host a WCF Service in WAS](#)

[How to: Host a WCF Service in a Managed Windows Service](#)

[How to: Host a WCF Service in a Managed Application](#)

How to: Host a WCF service in a managed app

9/18/2018 • 5 minutes to read • [Edit Online](#)

To host a service inside a managed application, embed the code for the service inside the managed application code, define an endpoint for the service either imperatively in code, declaratively through configuration, or using default endpoints, and then create an instance of [ServiceHost](#).

To start receiving messages, call [Open](#) on [ServiceHost](#). This creates and opens the listener for the service. Hosting a service in this way is often referred to as "self-hosting" because the managed application is doing the hosting work itself. To close the service, call [CommunicationObject.Close](#) on [ServiceHost](#).

A service can also be hosted in a managed Windows service, in Internet Information Services (IIS), or in Windows Process Activation Service (WAS). For more information about hosting options for a service, see [Hosting Services](#).

Hosting a service in a managed application is the most flexible option because it requires the least infrastructure to deploy. For more information about hosting services in managed applications, see [Hosting in a Managed Application](#).

The following procedure demonstrates how to implement a self-hosted service in a console application.

Create a self-hosted service

1. Create a new console application:
 - a. Open Visual Studio and select **New > Project** from the **File** menu.
 - b. In the **Installed Templates** list, select **Visual C#** or **Visual Basic**, and then select **Windows Desktop**.
 - c. Select the **Console App** template. Type `SelfHost` in the **Name** box and then choose **OK**.
2. Right-click **SelfHost** in **Solution Explorer** and select **Add Reference**. Select **System.ServiceModel** from the **.NET** tab and then choose **OK**.

TIP

If the **Solution Explorer** window is not visible, select **Solution Explorer** from the **View** menu.

3. Double-click **Program.cs** or **Module1.vb** in **Solution Explorer** to open it in the code window, if it's not already open. Add the following statements at the top of the file:

```
using System.ServiceModel;  
using System.ServiceModel.Description;
```

```
Imports System.ServiceModel  
Imports System.ServiceModel.Description
```

4. Define and implement a service contract. This example defines a `HelloWorldService` that returns a message based on the input to the service.

```
[ServiceContract]
public interface IHelloWorldService
{
    [OperationContract]
    string SayHello(string name);
}

public class HelloWorldService : IHelloWorldService
{
    public string SayHello(string name)
    {
        return string.Format("Hello, {0}", name);
    }
}
```

```
<ServiceContract(>>
Public Interface IHelloWorldService
    <OperationContract(>>
        Function SayHello(ByVal name As String) As String
    End Interface

Public Class HelloWorldService
    Implements IHelloWorldService

    Public Function SayHello(ByVal name As String) As String Implements IHelloWorldService.SayHello
        Return String.Format("Hello, {0}", name)
    End Function
End Class
```

NOTE

For more information about how to define and implement a service interface, see [How to: Define a Service Contract](#) and [How to: Implement a Service Contract](#).

- At the top of the `Main` method, create an instance of the `Uri` class with the base address for the service.

```
Uri baseAddress = new Uri("http://localhost:8080/hello");
```

```
Dim baseAddress As Uri = New Uri("http://localhost:8080/hello")
```

- Create an instance of the `ServiceHost` class, passing a `Type` that represents the service type and the base address Uniform Resource Identifier (URI) to the `ServiceHost(Type, Uri[])`. Enable metadata publishing, and then call the `Open` method on the `ServiceHost` to initialize the service and prepare it to receive messages.

```
// Create the ServiceHost.
using (ServiceHost host = new ServiceHost(typeof(HelloWorldService), baseAddress))
{
    // Enable metadata publishing.
    ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
    smb.HttpGetEnabled = true;
    smb.MetadataExporter.PolicyVersion = PolicyVersion.Policy15;
    host.Description.Behaviors.Add(smb);

    // Open the ServiceHost to start listening for messages. Since
    // no endpoints are explicitly configured, the runtime will create
    // one endpoint per base address for each service contract implemented
    // by the service.
    host.Open();

    Console.WriteLine("The service is ready at {0}", baseAddress);
    Console.WriteLine("Press <Enter> to stop the service.");
    Console.ReadLine();

    // Close the ServiceHost.
    host.Close();
}
```

```
' Create the ServiceHost.
Using host As New ServiceHost(GetType(HelloWorldService), baseAddress)

    ' Enable metadata publishing.
    Dim smb As New ServiceMetadataBehavior()
    smb.HttpGetEnabled = True
    smb.MetadataExporter.PolicyVersion = PolicyVersion.Policy15
    host.Description.Behaviors.Add(smb)

    ' Open the ServiceHost to start listening for messages. Since
    ' no endpoints are explicitly configured, the runtime will create
    ' one endpoint per base address for each service contract implemented
    ' by the service.
    host.Open()

    Console.WriteLine("The service is ready at {0}", baseAddress)
    Console.WriteLine("Press <Enter> to stop the service.")
    Console.ReadLine()

    ' Close the ServiceHost.
    host.Close()

End Using
```

NOTE

This example uses default endpoints, and no configuration file is required for this service. If no endpoints are configured, then the runtime creates one endpoint for each base address for each service contract implemented by the service. For more information about default endpoints, see [Simplified Configuration](#) and [Simplified Configuration for WCF Services](#).

7. Press **Ctrl+Shift+B** to build the solution.

Test the service

1. Press **Ctrl+F5** to run the service.
2. Open **WCF Test Client**.

TIP

To open **WCF Test Client**, open Developer Command Prompt for Visual Studio and execute **WcfTestClient.exe**.

3. Select **Add Service** from the **File** menu.
4. Type `http://localhost:8080/hello` into the address box and click **OK**.

TIP

Make sure the service is running or else this step fails. If you have changed the base address in the code, then use the modified base address in this step.

5. Double-click **SayHello** under the **My Service Projects** node. Type your name into the **Value** column in the **Request** list, and click **Invoke**.

A reply message appears in the **Response** list.

Example

The following example creates a `ServiceHost` object to host a service of type `HelloWorldService`, and then calls the `Open` method on `ServiceHost`. A base address is provided in code, metadata publishing is enabled, and default endpoints are used.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel;
using System.ServiceModel.Description;

namespace SelfHost
{
    [ServiceContract]
    public interface IHelloWorldService
    {
        [OperationContract]
        string SayHello(string name);
    }

    public class HelloWorldService : IHelloWorldService
    {
        public string SayHello(string name)
        {
            return string.Format("Hello, {0}", name);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Uri baseAddress = new Uri("http://localhost:8080/hello");

            // Create the ServiceHost.
            using (ServiceHost host = new ServiceHost(typeof(HelloWorldService), baseAddress))
            {
                // Enable metadata publishing.
                ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
                smb.HttpGetEnabled = true;
                smb.MetadataExporter.PolicyVersion = PolicyVersion.Policy15;
                host.Description.Behaviors.Add(smb);

                // Open the ServiceHost to start listening for messages. Since
                // no endpoints are explicitly configured, the runtime will create
                // one endpoint per base address for each service contract implemented
                // by the service.
                host.Open();

                Console.WriteLine("The service is ready at {0}", baseAddress);
                Console.WriteLine("Press <Enter> to stop the service.");
                Console.ReadLine();

                // Close the ServiceHost.
                host.Close();
            }
        }
    }
}

```

```
Imports System.ServiceModel
Imports System.ServiceModel.Description

Module Module1

    <ServiceContract(>>
    Public Interface IHelloWorldService
        <OperationContract(>>
            Function SayHello(ByVal name As String) As String
        End Interface

    Public Class HelloWorldService
        Implements IHelloWorldService

        Public Function SayHello(ByVal name As String) As String Implements IHelloWorldService.SayHello
            Return String.Format("Hello, {0}", name)
        End Function
    End Class

    Sub Main()
        Dim baseAddress As Uri = New Uri("http://localhost:8080/hello")

        ' Create the ServiceHost.
        Using host As New ServiceHost(GetType(HelloWorldService), baseAddress)

            ' Enable metadata publishing.
            Dim smb As New ServiceMetadataBehavior()
            smb.HttpGetEnabled = True
            smb.MetadataExporter.PolicyVersion = PolicyVersion.Policy15
            host.Description.Behaviors.Add(smb)

            ' Open the ServiceHost to start listening for messages. Since
            ' no endpoints are explicitly configured, the runtime will create
            ' one endpoint per base address for each service contract implemented
            ' by the service.
            host.Open()

            Console.WriteLine("The service is ready at {0}", baseAddress)
            Console.WriteLine("Press <Enter> to stop the service.")
            Console.ReadLine()

            ' Close the ServiceHost.
            host.Close()

        End Using

    End Sub

End Module
```

See also

- [Uri](#)
- [AppSettings](#)
- [ConfigurationManager](#)
- [How to: Host a WCF Service in IIS](#)
- [Self-Host](#)
- [Hosting Services](#)
- [How to: Define a Service Contract](#)
- [How to: Implement a Service Contract](#)
- [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#)

- [Using Bindings to Configure Services and Clients](#)
- [System-Provided Bindings](#)

How to: Host a WCF Service Written with .NET Framework 3.5 in IIS Running Under .NET Framework 4

5/4/2018 • 2 minutes to read • [Edit Online](#)

When hosting a Windows Communication Foundation (WCF) service written with .NET Framework version 3.5 on a machine running .NET Framework version 4, you may get a [ProtocolException](#) with the following text.

```
Unhandled Exception: System.ServiceModel.ProtocolException: The content type text/html; charset=utf-8 of the response message does not match the content type of the binding (application/soap+xml; charset=utf-8). If using a custom encoder, be sure that the IsContentTypeSupported method is implemented properly. The first 1024 bytes of the response were: '<html>    <head>        <title>The application domain or application pool is currently running version 4.0 or later of the .NET Framework. This can occur if IIS settings have been set to 4.0 or later for this Web application, or if you are using version 4.0 or later of the ASP.NET Web Development Server. The <compilation> element in the Web.config file for this Web application does not contain the required 'targetFrameworkMoniker' attribute for this version of the .NET Framework (for example, '<compilation targetFrameworkMoniker=".NETFramework,Version=v4.0">'). Update the Web.config file with this attribute, or configure the Web application to use a different version of the .NET Framework.</title>...
```

Or if you try to browse to the service's .svc file you may see an error page with the following text.

```
The application domain or application pool is currently running version 4.0 or later of the .NET Framework. This can occur if IIS settings have been set to 4.0 or later for this Web application, or if you are using version 4.0 or later of the ASP.NET Web Development Server. The <compilation> element in the Web.config file for this Web application does not contain the required 'targetFrameworkMoniker' attribute for this version of the .NET Framework (for example, '<compilation targetFrameworkMoniker=".NETFramework,Version=v4.0">'). Update the Web.config file with this attribute, or configure the Web application to use a different version of the .NET Framework.
```

These errors occur because the application domain IIS is running within is running .NET Framework 4 and the WCF service is expecting to run under .NET Framework 3.5. This topic explains the modifications required to get the service to run.

Next find the `<compilers>` element and change the `CompilerVersion` provider option to have a value of 4.0. By default, there are two `<compiler>` elements under the `<compilers>` element. You must update the `CompilerVersion` provider option for both as shown in the following example.

```

<system.codedom>
  <compilers>
    <compiler language="c#;cs;csharp" extension=".cs" warningLevel="4"
      type="Microsoft.CSharp.CSharpCodeProvider, System, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089">
      <providerOption name="CompilerVersion" value="v3.5"/>
      <providerOption name="WarnAsError" value="false"/>
    </compiler>
    <compiler language="vb;vbs;visualbasic;vbscript" extension=".vb" warningLevel="4"
      type="Microsoft.VisualBasic.VBCodeProvider, System, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089">
      <providerOption name="CompilerVersion" value="v3.5"/>
      <providerOption name="OptionInfer" value="true"/>
      <providerOption name="WarnAsError" value="false"/>
    </compiler>
  </compilers>
</system.codedom>

```

Add the required targetFramework attribute

1. Open the service's Web.config file and look for the `< compilation >` element.
2. Add the `targetFramework` attribute to the `< compilation >` element as shown in the following example.

```

<compilation debug="false"
  targetFramework="4.0">

  <assemblies>
    <add assembly="System.Core, Version=3.5.0.0, Culture=neutral,
    PublicKeyToken=B77A5C561934E089"/>
    <add assembly="System.Xml.Linq, Version=3.5.0.0, Culture=neutral,
    PublicKeyToken=B77A5C561934E089"/>
    <add assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
    PublicKeyToken=31BF3856AD364E35"/>
    <add assembly="System.Data.DataSetExtensions, Version=3.5.0.0, Culture=neutral,
    PublicKeyToken=B77A5C561934E089"/>
  </assemblies>

</compilation>

```

3. Find the `< compilers >` element and change the CompilerVersion provider option to have a value of 4.0. By default, there are two `< compiler >` elements under the `< compilers >` element. You must update the CompilerVersion provider option for both as shown in the following example.

```

<system.codedom>
  <compilers>
    <compiler language="c#;cs;csharp" extension=".cs" warningLevel="4"
      type="Microsoft.CSharp.CSharpCodeProvider, System, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089">
      <providerOption name="CompilerVersion" value="v3.5"/>
      <providerOption name="WarnAsError" value="false"/>
    </compiler>
    <compiler language="vb;vbs;visualbasic;vbscript" extension=".vb" warningLevel="4"
      type="Microsoft.VisualBasic.VBCodeProvider, System, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089">
      <providerOption name="CompilerVersion" value="v3.5"/>
      <providerOption name="OptionInfer" value="true"/>
      <providerOption name="WarnAsError" value="false"/>
    </compiler>
  </compilers>
</system.codedom>

```

Building Clients

5/5/2018 • 2 minutes to read • [Edit Online](#)

The topics in this section demonstrate how to create and use a Windows Communication Foundation (WCF) client.

In This Section

[WCF Client Overview](#)

Provides an overview of WCF clients and how they work.

[Accessing Services Using a WCF Client](#)

Outlines the basic method of creating a WCF client object for use in a client application.

[Securing Clients](#)

Describes how to configure security for WCF clients.

Reference

[ClientBase<TChannel>](#)

[DuplexClientBase<TChannel>](#)

[DuplexChannelFactory<TChannel>](#)

[ServiceContractAttribute](#)

[OperationContractAttribute](#)

[DataContractAttribute](#)

[DataMemberAttribute](#)

Related Sections

[Hosting Services](#)

[Designing and Implementing Services](#)

WCF Client Overview

11/13/2018 • 9 minutes to read • [Edit Online](#)

This section describes what client applications do, how to configure, create, and use a Windows Communication Foundation (WCF) client, and how to secure client applications.

Using WCF Client Objects

A client application is a managed application that uses a WCF client to communicate with another application. To create a client application for a WCF service requires the following steps:

1. Obtain the service contract, bindings, and address information for a service endpoint.
2. Create a WCF client using that information.
3. Call operations.
4. Close the WCF client object.

The following sections discuss these steps and provide brief introductions to the following issues:

- Handling errors.
- Configuring and securing clients.
- Creating callback objects for duplex services.
- Calling services asynchronously.
- Calling services using client channels.

Obtain the Service Contract, Bindings, and Addresses

In WCF, services and clients model contracts using managed attributes, interfaces, and methods. To connect to a service in a client application, you need to obtain the type information for the service contract. Typically, you do this by using the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#), which downloads metadata from the service, converts it to a managed source code file in the language of your choice, and creates a client application configuration file that you can use to configure your WCF client object. For example, if you are going to create an WCF client object to invoke a `MyCalculatorService`, and you know that the metadata for that service is published at `http://computerName/MyCalculatorService/Service.svc?wsdl`, then the following code example shows how to use Svcutil.exe to obtain a `ClientCode.vb` file that contains the service contract in managed code.

```
svcutil /language:vb /out:ClientCode.vb /config:app.config  
http://computerName/MyCalculatorService/Service.svc?wsdl
```

You can either compile this contract code into the client application or into another assembly that the client application can then use to create an WCF client object. You can use the configuration file to configure the client object to properly connect to the service .

For an example of this process, see [How to: Create a Client](#). For more complete information about contracts, see [Contracts](#).

Create a WCF Client Object

A WCF client is a local object that represents a WCF service in a form that the client can use to communicate with the remote service. WCF client types implement the target service contract, so when you create one and configure it, you can then use the client object directly to invoke service operations. The WCF run time converts the method calls into messages, sends them to the service, listens for the reply, and returns those values to the WCF client object as return values or `out` or `ref` parameters.

You can also use WCF client channel objects to connect with and use services. For details, see [WCF Client Architecture](#).

Creating a New WCF Object

To illustrate the use of a `ClientBase<TChannel>` class, assume the following simple service contract has been generated from a service application.

NOTE

If you are using Visual Studio to create your WCF client, objects are loaded automatically into the object browser when you add a service reference to your project.

```
[System.ServiceModel.ServiceContractAttribute(
    Namespace = "http://microsoft.wcf.documentation"
)]
public interface ISampleService
{
    [System.ServiceModel.OperationContractAttribute(
        Action = "http://microsoft.wcf.documentation/ISampleService/SampleMethod",
        ReplyAction = "http://microsoft.wcf.documentation/ISampleService/SampleMethodResponse"
    )]
    [System.ServiceModel.FaultContractAttribute(
        typeof(microsoft.wcf.documentation.SampleFault),
        Action = "http://microsoft.wcf.documentation/ISampleService/SampleMethodSampleFaultFault"
    )]
    string SampleMethod(string msg);
}
```

If you are not using Visual Studio, examine the generated contract code to find the type that extends `ClientBase<TChannel>` and the service contract interface `ISampleService`. In this case, that type looks like the following code:

```
[System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")]
public partial class SampleServiceClient : System.ServiceModel.ClientBase<ISampleService>, ISampleService
{
    public SampleServiceClient()
    {
    }

    public SampleServiceClient(string endpointConfigurationName)
        :
            base(endpointConfigurationName)
    {
    }

    public SampleServiceClient(string endpointConfigurationName, string remoteAddress)
        :
            base(endpointConfigurationName, remoteAddress)
    {
    }

    public SampleServiceClient(string endpointConfigurationName, System.ServiceModel.EndpointAddress
remoteAddress)
        :
            base(endpointConfigurationName, remoteAddress)
    {
    }

    public SampleServiceClient(System.ServiceModel.Channels.Binding binding,
System.ServiceModel.EndpointAddress remoteAddress)
        :
            base(binding, remoteAddress)
    {
    }
    public string SampleMethod(string msg)
    {
        return base.Channel.SampleMethod(msg);
    }
}
```

This class can be created as a local object using one of the constructors, configured, and then used to connect to a service of the type `ISampleService`.

It is recommended that you create your WCF client object first, and then use it and close it inside a single try/catch block. You should not use the `using` statement (`Using` in Visual Basic) because it may mask exceptions in certain failure modes. For more information, see the following sections as well as [Use Close and Abort to release WCF client resources](#).

Contracts, Bindings, and Addresses

Before you can create a WCF client object, you must configure the client object. Specifically, it must have a service *endpoint* to use. An endpoint is the combination of a service contract, a binding, and an address. (For more information about endpoints, see [Endpoints: Addresses, Bindings, and Contracts](#).) Typically, this information is located in the `<endpoint>` element in a client application configuration file, such as the one the Svcutil.exe tool generates, and is loaded automatically when you create your client object. Both WCF client types also have overloads that enable you to programmatically specify this information.

For example, a generated configuration file for an `ISampleService` used in the preceding examples contains the following endpoint information.

```

<configuration>
  <system.serviceModel>
    <bindings>
      <wsHttpBinding>
        <binding name="WSHttpBinding_ISampleService" closeTimeout="00:01:00"
          openTimeout="00:01:00" receiveTimeout="00:01:00" sendTimeout="00:01:00"
          bypassProxyOnLocal="false" transactionFlow="false" hostNameComparisonMode="StrongWildcard"
          maxBufferPoolSize="524288" maxReceivedMessageSize="65536"
          messageEncoding="Text" textEncoding="utf-8" useDefaultWebProxy="true"
          allowCookies="false">
          <readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
            maxBytesPerRead="4096" maxNameTableCharCount="16384" />
          <reliableSession ordered="true" inactivityTimeout="00:10:00"
            enabled="false" />
          <security mode="Message">
            <transport clientCredentialType="None" proxyCredentialType="None"
              realm="" />
            <message clientCredentialType="Windows" negotiateServiceCredential="true"
              algorithmSuite="Default" establishSecurityContext="true" />
          </security>
        </binding>
      </wsHttpBinding>
    </bindings>
    <client>
      <endpoint address="http://localhost:8080/SampleService" binding="wsHttpBinding"
        bindingConfiguration="WSHttpBinding_ISampleService" contract="ISampleService"
        name="WSHttpBinding_ISampleService">
      </endpoint>
    </client>
  </system.serviceModel>
</configuration>

```

This configuration file specifies a target endpoint in the `<client>` element. For more information about using multiple target endpoints, see the [ClientBase<TChannel>.ClientBase<TChannel>](#) or the [ChannelFactory<TChannel>.ChannelFactory<TChannel>](#) constructors.

Calling Operations

Once you have a client object created and configured, create a try/catch block, call operations in the same way that you would if the object were local, and close the WCF client object. When the client application calls the first operation, WCF automatically opens the underlying channel, and the underlying channel is closed when the object is recycled. (Alternatively, you can also explicitly open and close the channel prior to or subsequent to calling other operations.)

For example, if you have the following service contract:

```

namespace Microsoft.ServiceModel.Samples
{
    using System;
    using System.ServiceModel;

    [ServiceContract(Namespace = "http://Microsoft.ServiceModel.Samples")]
    public interface ICalculator
    {
        [OperationContract]
        double Add(double n1, double n2);
        [OperationContract]
        double Subtract(double n1, double n2);
        [OperationContract]
        double Multiply(double n1, double n2);
        [OperationContract]
        double Divide(double n1, double n2);
    }
}

```

```

Namespace Microsoft.ServiceModel.Samples

Imports System
Imports System.ServiceModel

<ServiceContract(Namespace:= _
"http://Microsoft.ServiceModel.Samples")> _
Public Interface ICalculator
    <OperationContract> _
    Function Add(n1 As Double, n2 As Double) As Double
    <OperationContract> _
    Function Subtract(n1 As Double, n2 As Double) As Double
    <OperationContract> _
    Function Multiply(n1 As Double, n2 As Double) As Double
    <OperationContract> _
    Function Divide(n1 As Double, n2 As Double) As Double
End Interface

```

You can call operations by creating a WCF client object and calling its methods, as the following code example demonstrates. Note that the opening, calling, and closing of the WCF client object occurs within a single try/catch block. For more information, see [Accessing Services Using a WCF Client](#) and [Use Close and Abort to release WCF client resources](#).

```

CalculatorClient wcfClient = new CalculatorClient();
try
{
    Console.WriteLine(wcfClient.Add(4, 6));
    wcfClient.Close();
}
catch (TimeoutException timeout)
{
    // Handle the timeout exception.
    wcfClient.Abort();
}
catch (CommunicationException commException)
{
    // Handle the communication exception.
    wcfClient.Abort();
}

```

Handling Errors

Exceptions can occur in a client application when opening the underlying client channel (whether explicitly or automatically by calling an operation), using the client or channel object to call operations, or when closing the underlying client channel. It is recommended at a minimum that applications expect to handle possible [System.TimeoutException](#) and [System.ServiceModel.CommunicationException](#) exceptions in addition to any [System.ServiceModel.FaultException](#) objects thrown as a result of SOAP faults returned by operations. SOAP faults specified in the operation contract are raised to client applications as a [System.ServiceModel.FaultException<TDetail>](#) where the type parameter is the detail type of the SOAP fault. For more information about handling error conditions in a client application, see [Sending and Receiving Faults](#). For a complete sample the shows how to handle errors in a client, see [Expected Exceptions](#).

Configuring and Securing Clients

Configuring a client starts with the required loading of target endpoint information for the client or channel object, usually from a configuration file, although you can also load this information programmatically using the client constructors and properties. However, additional configuration steps are required to enable certain client behavior and for many security scenarios.

For example, security requirements for service contracts are declared in the service contract interface, and if Svcutil.exe created a configuration file, that file usually contains a binding that is capable of supporting the security requirements of the service. In some cases, however, more security configuration may be required, such as configuring client credentials. For complete information about the configuration of security for WCF clients, see [Securing Clients](#).

In addition, some custom modifications can be enabled in client applications, such as custom run-time behaviors. For more information about how to configure a custom client behavior, see [Configuring Client Behaviors](#).

Creating Callback Objects for Duplex Services

Duplex services specify a callback contract that the client application must implement in order to provide a callback object for the service to call according to the requirements of the contract. Although callback objects are not full services (for example, you cannot initiate a channel with a callback object), for the purposes of implementation and configuration they can be thought of as a kind of service.

Clients of duplex services must:

- Implement a callback contract class.
- Create an instance of the callback contract implementation class and use it to create the [System.ServiceModel.InstanceContext](#) object that you pass to the WCF client constructor.
- Invoke operations and handle operation callbacks.

Duplex WCF client objects function like their nonduplex counterparts, with the exception that they expose the functionality necessary to support callbacks, including the configuration of the callback service.

For example, you can control various aspects of callback object runtime behavior by using properties of the [System.ServiceModel.CallbackBehaviorAttribute](#) attribute on the callback class. Another example is the use of the [System.ServiceModel.Description.CallbackDebugBehavior](#) class to enable the return of exception information to services that call the callback object. For more information, see [Duplex Services](#). For a complete sample, see [Duplex](#).

On Windows XP computers running Internet Information Services (IIS) 5.1, duplex clients must specify a client base address using the [System.ServiceModel.WSDualHttpBinding](#) class or an exception is thrown. The following code example shows how to do this in code.

```
WSDualHttpBinding dualBinding = new WSDualHttpBinding();
EndpointAddress endptadr = new EndpointAddress("http://localhost:12000/DuplexTestUsingCode/Server");
dualBinding.ClientBaseAddress = new Uri("http://localhost:8000/DuplexTestUsingCode/Client/");
```

```
Dim dualBinding As New WSDualHttpBinding()
Dim endptadr As New EndpointAddress("http://localhost:12000/DuplexTestUsingCode/Server")
dualBinding.ClientBaseAddress = New Uri("http://localhost:8000/DuplexTestUsingCode/Client/")
```

The following code shows how to do this in a configuration file

```
<client>
  <endpoint
    name = "ServerEndpoint"
    address="http://localhost:12000/DuplexUsingConfig/Server"
    bindingConfiguration="WSDualHttpBinding_IDuplex"
    binding="wsDualHttpBinding"
    contract="IDuplex"
  />
</client>
<bindings>
  <wsDualHttpBinding>
    <binding
      name="WSDualHttpBinding_IDuplex"
      clientBaseAddress="http://localhost:8000/myClient/"
    />
  </wsDualHttpBinding>
</bindings>
```

Calling Services Asynchronously

How operations are called is entirely up to the client developer. This is because the messages that make up an operation can be mapped to either synchronous or asynchronous methods when expressed in managed code. Therefore, if you want to build a client that calls operations asynchronously, you can use Svcutil.exe to generate asynchronous client code using the `/async` option. For more information, see [How to: Call Service Operations Asynchronously](#).

Calling Services Using WCF Client Channels

WCF client types extend `ClientBase<TChannel>`, which itself derives from `System.ServiceModel.IClientChannel` interface to expose the underlying channel system. You can invoke services by using the target service contract with the `System.ServiceModel.ChannelFactory<TChannel>` class. For details, see [WCF Client Architecture](#).

See Also

[System.ServiceModel.ClientBase<TChannel>](#)

[System.ServiceModel.ChannelFactory<TChannel>](#)

Accessing Services Using a WCF Client

8/28/2018 • 4 minutes to read • [Edit Online](#)

After you create a service, the next step is to create a WCF client proxy. A client application uses the WCF client proxy to communicate with the service. Client applications usually import a service's metadata to generate WCF client code that can be used to invoke the service.

The basic steps for creating a WCF client include the following:

1. Compile the service code.
2. Generate the WCF client proxy.
3. Instantiate the WCF client proxy.

The WCF client proxy can be generated manually by using the Service Model Metadata Utility Tool (SvcUtil.exe) for more information see, [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#). The WCF client proxy can also be generated within Visual Studio using the **Add Service Reference** feature. To generate the WCF client proxy using either method the service must be running. If the service is self-hosted you must run the host. If the service is hosted in IIS/WAS you do not need to do anything else.

ServiceModel Metadata Utility Tool

The [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) is a command-line tool for generating code from metadata. The following use is an example of a basic Svcutil.exe command.

```
Svcutil.exe <service's Metadata Exchange (MEX) address or HTTP GET address>
```

Alternatively, you can use Svcutil.exe with Web Services Description Language (WSDL) and XML Schema definition language (XSD) files on the file system.

```
Svcutil.exe <list of WSDL and XSD files on file system>
```

The result is a code file that contains WCF client code that the client application can use to invoke the service.

You can also use the tool to generate configuration files.

```
Svcutil.exe <file1 [,file2]>
```

If only one file name is given, that is the name of the output file. If two file names are given, then the first file is an input configuration file whose contents are merged with the generated configuration and written out into the second file. For more information about configuration, see [Configuring Bindings for Services](#).

IMPORTANT

Unsecured metadata requests pose certain risks in the same way that any unsecured network request does: If you are not certain that the endpoint you are communicating with is who it says it is, the information you retrieve might be metadata from a malicious service.

Add Service Reference in Visual Studio

With the service running, right click the project that will contain the WCF client proxy and select **Add > Service Reference**. In the **Add Service Reference Dialog**, type in the URL to the service you want to call and click the **Go** button. The dialog will display a list of services available at the address you specify. Double click the service to see the contracts and operations available, specify a namespace for the generated code, and click the **OK** button.

Example

The following code example shows a service contract created for a service.

```
// Define a service contract.
[ServiceContract(Namespace="http://Microsoft.ServiceModel.Samples")]
public interface ICalculator
{
    [OperationContract]
    double Add(double n1, double n2);
    // Other methods are not shown here.
}
```

```
' Define a service contract.
<ServiceContract(Namespace="http://Microsoft.ServiceModel.Samples")> _
Public Interface ICalculator
    <OperationContract()> _
    Function Add(ByVal n1 As Double, ByVal n2 As Double) As Double
    ' Other methods are not shown here.
End Interface
```

The ServiceModel Metadata utility tool and **Add Service Reference** in Visual Studio generates the following WCF client class. The class inherits from the generic [ClientBase<TChannel>](#) class and implements the `ICalculator` interface. The tool also generates the `ICalculator` interface (not shown here).

```
public partial class CalculatorClient : System.ServiceModel.ClientBase<ICalculator>, ICalculator
{
    public CalculatorClient()
    {}

    public CalculatorClient(string endpointConfigurationName) :
        base(endpointConfigurationName)
    {}

    public CalculatorClient(string endpointConfigurationName, string remoteAddress) :
        base(endpointConfigurationName, remoteAddress)
    {}

    public CalculatorClient(string endpointConfigurationName,
        System.ServiceModel.EndpointAddress remoteAddress) :
        base(endpointConfigurationName, remoteAddress)
    {}

    public CalculatorClient(System.ServiceModel.Channels.Binding binding,
        System.ServiceModel.EndpointAddress remoteAddress) :
        base(binding, remoteAddress)
    {}

    public double Add(double n1, double n2)
    {
        return base.Channel.Add(n1, n2);
    }
}
```

```

Partial Public Class CalculatorClient
    Inherits System.ServiceModel.ClientBase(Of ICalculator)
    Implements ICalculator

    Public Sub New()
        MyBase.New
    End Sub

    Public Sub New(ByVal endpointConfigurationName As String)
        MyBase.New(endpointConfigurationName)
    End Sub

    Public Sub New(ByVal endpointConfigurationName As String, ByVal remoteAddress As String)
        MyBase.New(endpointConfigurationName, remoteAddress)
    End Sub

    Public Sub New(ByVal endpointConfigurationName As String,
        ByVal remoteAddress As System.ServiceModel.EndpointAddress)
        MyBase.New(endpointConfigurationName, remoteAddress)
    End Sub

    Public Sub New(ByVal binding As System.ServiceModel.Channels.Binding,
        ByVal remoteAddress As System.ServiceModel.EndpointAddress)
        MyBase.New(binding, remoteAddress)
    End Sub

    Public Function Add(ByVal n1 As Double, ByVal n2 As Double) As Double
        Implements ICalculator.Add
        Return MyBase.Channel.Add(n1, n2)
    End Function
End Class

```

Using the WCF Client

To use the WCF client, create an instance of the WCF client, and then call its methods, as shown in the following code.

```

// Create a client object with the given client endpoint configuration.
CalculatorClient calcClient = new CalculatorClient("CalculatorEndpoint");
// Call the Add service operation.
double value1 = 100.00D;
double value2 = 15.99D;
double result = calcClient.Add(value1, value2);
Console.WriteLine("Add({0},{1}) = {2}", value1, value2, result);

```

```

' Create a client object with the given client endpoint configuration.
Dim calcClient As CalculatorClient = _
    New CalculatorClient("CalculatorEndpoint")

' Call the Add service operation.
Dim value1 As Double = 100.00D
Dim value2 As Double = 15.99D
Dim result As Double = calcClient.Add(value1, value2)
Console.WriteLine("Add({0},{1}) = {2}", value1, value2, result)

```

Debugging Exceptions Thrown by a Client

Many exceptions thrown by a WCF client are caused by an exception on the service. Some examples of this are:

- [SocketException](#): An existing connection was forcibly closed by the remote host.

- [CommunicationException](#): The underlying connection was closed unexpectedly.
- [CommunicationObjectAbortedException](#): The socket connection was aborted. This could be caused by an error processing your message, a receive time-out being exceeded by the remote host, or an underlying network resource issue.

When these types of exceptions occur, the best way to solve the problem is to turn on tracing on the service side and determine what exception occurred there. For more information about tracing, see [Tracing](#) and [Using Tracing to Troubleshoot Your Application](#).

See Also

- [How to: Create a Client](#)
- [How to: Access Services with a Duplex Contract](#)
- [How to: Call Service Operations Asynchronously](#)
- [How to: Access Services with One-Way and Request-Reply Contracts](#)
- [How to: Access a WSE 3.0 Service](#)
- [Understanding Generated Client Code](#)
- [How to: Improve the Startup Time of WCF Client Applications using the XmlSerializer](#)
- [Specifying Client Run-Time Behavior](#)
- [Configuring Client Behaviors](#)

Add Service Reference in a Portable Subset Project

8/31/2018 • 2 minutes to read • [Edit Online](#)

Portable subset projects enable .NET assembly programmers to maintain a single source tree and build system while still supporting multiple .NET implementations (desktop, Silverlight, Windows Phone, and XBOX). Portable subset projects only reference .NET portable libraries which are a .NET framework assembly that can be used on any .NET implementation.

Add Service Reference Details

When adding a service reference in a portable subset project the following restrictions are enforced:

1. For [XmlSerializer](#), only literal encodings are allowed. SOAP encodings generate an error during import.
2. For services that use [DataContractSerializer](#) scenarios, a data contract surrogate is provided to ensure that reused types come only from the portable subset.
3. Endpoints which rely on bindings not supported in portable libraries (all bindings except [BasicHttpBinding](#), [WSHttpBinding](#) without transaction flow, reliable sessions, or MTOM encoding, and equivalent custom bindings) are ignored.
4. Message headers are deleted from all message descriptions in all operations before import.
5. Non-portable attributes [DesignerCategoryAttribute](#), [SerializableAttribute](#), and [TransactionFlowAttribute](#) are removed from generated client proxy code.
6. Non-portable properties [ProtectionLevel](#), [SessionMode](#), [IsInitiating](#), [IsTerminating](#) are removed from [ServiceContractAttribute](#), [OperationContractAttribute](#), and [FaultContractAttribute](#).
7. All service operations are generated as asynchronous operations on the client proxy.
8. Generated client constructors which use non-portable types are removed.
9. A [CookieContainer](#) instance is exposed on the generated client.
10. A comment is inserted at the top of the file identifying the assembly and version of the code generator:

```
// This code was auto-generated by Microsoft.VisualStudio.Portable.AddServiceReference, version 1.0.0.0
```
11. The [ISerializable](#) interface is not supported.
12. Net.Tcp and PollingDuplex bindings are not supported
13. The [DataContractSerializer](#) will always be used for faults.
14. [IsWrapped](#) is not supported in portable subset projects.

See Also

[Accessing Services Using a WCF Client Portable Class Library](#)

Specifying Client Run-Time Behavior

5/5/2018 • 3 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) clients, like Windows Communication Foundation (WCF) services, can be configured to modify the run-time behavior to suit the client application. Three attributes are available for specifying client run-time behavior. Duplex client callback objects can use the [CallbackBehaviorAttribute](#) and [CallbackDebugBehavior](#) attributes to modify their run-time behavior. The other attribute, [ClientViaBehavior](#), can be used to separate the logical destination from the immediate network destination. In addition, duplex client callback types can use some of the service-side behaviors. For more information, see [Specifying Service Run-Time Behavior](#).

Using the CallbackBehaviorAttribute

You can configure or extend the execution behavior of a callback contract implementation in a client application by using the [CallbackBehaviorAttribute](#) class. This attribute performs a similar function for the callback class as the [ServiceBehaviorAttribute](#) class, with the exception of instancing behavior and transaction settings.

The [CallbackBehaviorAttribute](#) class must be applied to the class that implements the callback contract. If applied to a nonduplex contract implementation, an [InvalidOperationException](#) exception is thrown at run time. The following code example shows a [CallbackBehaviorAttribute](#) class on a callback object that uses the [SynchronizationContext](#) object to determine the thread to marshal to, the [ValidateMustUnderstand](#) property to enforce message validation, and the [IncludeExceptionDetailInFaults](#) property to return exceptions as [FaultException](#) objects to the service for debugging purposes.

```
using System;
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.Threading;

namespace Microsoft.WCF.Documentation
{
    [CallbackBehaviorAttribute(
        IncludeExceptionDetailInFaults= true,
        UseSynchronizationContext=true,
        ValidateMustUnderstand=true
    )]
    public class Client : SampleDuplexHelloCallback
    {
        AutoResetEvent waitHandle;

        public Client()
        {
            waitHandle = new AutoResetEvent(false);
        }

        public void Run()
        {
            // Picks up configuration from the configuration file.
            SampleDuplexHelloClient wcClient
                = new SampleDuplexHelloClient(new InstanceContext(this), "WSDualHttpBinding_SampleDuplexHello");
            try
            {
                Console.ForegroundColor = ConsoleColor.White;
                Console.WriteLine("Enter a greeting to send and press ENTER: ");
                Console.Write(">>> ");
                Console.ForegroundColor = ConsoleColor.Green;
                string greeting = Console.ReadLine();
            }
        }
    }
}
```



```

        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("Called service with: \r\n\t" + greeting);
        wcfClient.Hello(greeting);
        Console.WriteLine("Execution passes service call and moves to the WaitHandle.");
        this.waitHandle.WaitOne();
        Console.ForegroundColor = ConsoleColor.Blue;
        Console.WriteLine("Set was called.");
        Console.Write("Press ");
        Console.ForegroundColor = ConsoleColor.Red;
        Console.Write("ENTER");
        Console.ForegroundColor = ConsoleColor.Blue;
        Console.Write(" to exit...");
        Console.ReadLine();
    }
    catch (TimeoutException timeProblem)
    {
        Console.WriteLine("The service operation timed out. " + timeProblem.Message);
        Console.ReadLine();
    }
    catch (CommunicationException commProblem)
    {
        Console.WriteLine("There was a communication problem. " + commProblem.Message);
        Console.ReadLine();
    }
}

public static void Main()
{
    Client client = new Client();
    client.Run();
}

public void Reply(string response)
{
    Console.WriteLine("Received output.");
    Console.WriteLine("\r\n\t" + response);
    this.waitHandle.Set();
}
}
}

```

```

Imports Microsoft.VisualBasic
Imports System
Imports System.ServiceModel
Imports System.ServiceModel.Channels
Imports System.Threading

Namespace Microsoft.WCF.Documentation
    <CallbackBehaviorAttribute(IncludeExceptionDetailInFaults:= True, UseSynchronizationContext:=True,
ValidateMustUnderstand:=True)> _
    Public Class Client
        Implements SampleDuplexHelloCallback
        Private waitHandle As AutoResetEvent

        Public Sub New()
            waitHandle = New AutoResetEvent(False)
        End Sub

        Public Sub Run()
            ' Picks up configuration from the configuration file.
            Dim wcfClient As New SampleDuplexHelloClient(New InstanceContext(Me),
"WSDualHttpBinding_SampleDuplexHello")
            Try
                Console.ForegroundColor = ConsoleColor.White
                Console.WriteLine("Enter a greeting to send and press ENTER: ")
                Console.Write(">>> ")
                Console.ForegroundColor = ConsoleColor.Green
                Dim greeting As String = Console.ReadLine()
                Console.ForegroundColor = ConsoleColor.White
                Console.WriteLine("Called service with: " & Constants.vbCrLf & Constants.vbTab & greeting)
                wcfClient.Hello(greeting)
                Console.WriteLine("Execution passes service call and moves to the WaitHandle.")
                Me.waitHandle.WaitOne()
                Console.ForegroundColor = ConsoleColor.Blue
                Console.WriteLine("Set was called.")
                Console.Write("Press ")
                Console.ForegroundColor = ConsoleColor.Red
                Console.Write("ENTER")
                Console.ForegroundColor = ConsoleColor.Blue
                Console.Write(" to exit...")
                Console.ReadLine()
            Catch timeProblem As TimeoutException
                Console.WriteLine("The service operation timed out. " & timeProblem.Message)
                Console.ReadLine()
            Catch commProblem As CommunicationException
                Console.WriteLine("There was a communication problem. " & commProblem.Message)
                Console.ReadLine()
            End Try
        End Sub

        Public Shared Sub Main()
            Dim client As New Client()
            client.Run()
        End Sub

        Public Sub Reply(ByVal response As String) Implements SampleDuplexHelloCallback.Reply
            Console.WriteLine("Received output.")
            Console.WriteLine(Constants.vbCrLf & Constants.vbTab & response)
            Me.waitHandle.Set()
        End Sub
    End Class
End Namespace

```

Using CallbackDebugBehavior to Enable the Flow of Managed Exception Information

You can enable the flow of managed exception information in a client callback object back to the service for debugging purposes by setting the [IncludeExceptionDetailInFaults](#) property to `true` either programmatically or from an application configuration file.

Returning managed exception information to services can be a security risk because exception details expose information about the internal client implementation that unauthorized services could use. In addition, although the [CallbackDebugBehavior](#) properties can also be set programmatically, it can be easy to forget to disable [IncludeExceptionDetailInFaults](#) when deploying.

Because of the security issues involved, it is strongly recommended that:

- You use an application configuration file to set the value of the [IncludeExceptionDetailInFaults](#) property to `true`.
- You do so only in controlled debugging scenarios.

The following code example shows a client configuration file that instructs WCF to return managed exception information from a client callback object in SOAP messages.

```
<client>
  <endpoint
    address="http://localhost:8080/DuplexHello"
    binding="wsDualHttpBinding"
    bindingConfiguration="WSDualHttpBinding_SampleDuplexHello"
    contract="SampleDuplexHello"
    name="WSDualHttpBinding_SampleDuplexHello"
    behaviorConfiguration="enableCallbackDebug">
  </endpoint>
</client>
<behaviors>
  <endpointBehaviors>
    <behavior name="enableCallbackDebug">
      <callbackDebug includeExceptionDetailInFaults="true"/>
    </behavior>
  </endpointBehaviors>
</behaviors>
```

Using the ClientViaBehavior Behavior

You can use the [ClientViaBehavior](#) behavior to specify the Uniform Resource Identifier for which the transport channel should be created. Use this behavior when the immediate network destination is not the intended processor of the message. This enables multiple-hop conversations when the calling application does not necessarily know the ultimate destination or when the destination `via` header is not an address.

See Also

[Specifying Service Run-Time Behavior](#)

Configuring Client Behaviors

5/4/2018 • 2 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) configures behaviors in two ways: either by referring to behavior configurations -- which are defined in the `<behavior>` section of a client application configuration file – or programmatically in the calling application. This topic describes both approaches.

When using a configuration file, behavior configuration is a named collection of configuration settings. The name of each behavior configuration must be unique. This string is used in the `behaviorConfiguration` attribute of an endpoint configuration to link the endpoint to the behavior.

Example

The following configuration code defines a behavior called `myBehavior`. The client endpoint references this behavior in the `behaviorConfiguration` attribute.

```
<configuration>
  <system.serviceModel>
    <behaviors>
      <endpointBehaviors>
        <behavior name="myBehavior">
          <clientVia />
        </behavior>
      </endpointBehaviors>
    </behaviors>
    <bindings>
      <basicHttpBinding>
        <binding name="myBinding" maxReceivedMessageSize="10000" />
      </basicHttpBinding>
    </bindings>
    <client>
      <endpoint address="myAddress" binding="basicHttpBinding" bindingConfiguration="myBinding"
behaviorConfiguration="myBehavior" contract="myContract" />
    </client>
  </system.serviceModel>
</configuration>
```

Using Behaviors Programmatically

You can also configure or insert behaviors programmatically by locating the appropriate `Behaviors` property on the Windows Communication Foundation (WCF) client object or on the client channel factory object prior to opening the client.

Example

The following code example shows how to programmatically insert a behavior by accessing the `Behaviors` property on the `ServiceEndpoint` returned from the `Endpoint` property prior to the creation of the channel object.

```

public class Client
{
    public static void Main()
    {
        try
        {
            // Picks up configuration from the config file.
            ChannelFactory<ISampleServiceChannel> factory
                = new ChannelFactory<ISampleServiceChannel>("WSHttpBinding_ISampleService");

            // Add the client side behavior programmatically to all created channels.
            factory.Endpoint.Behaviors.Add(new EndpointBehaviorMessageInspector());

            ISampleServiceChannel wcfClientChannel = factory.CreateChannel();

            // Making calls.
            Console.WriteLine("Enter the greeting to send: ");
            string greeting = Console.ReadLine();
            Console.WriteLine("The service responded: " + wcfClientChannel.SampleMethod(greeting));

            Console.WriteLine("Press ENTER to exit:");
            Console.ReadLine();

            // Done with service.
            wcfClientChannel.Close();
            Console.WriteLine("Done!");
        }
        catch (TimeoutException timeProblem)
        {
            Console.WriteLine("The service operation timed out. " + timeProblem.Message);
            Console.Read();
        }
        catch (FaultException<SampleFault> fault)
        {
            Console.WriteLine("SampleFault fault occurred: {0}", fault.Detail.FaultMessage);
            Console.Read();
        }
        catch (CommunicationException commProblem)
        {
            Console.WriteLine("There was a communication problem. " + commProblem.Message);
            Console.Read();
        }
    }
}

```

```

Public Class Client
    Public Shared Sub Main()
        Try
            ' Picks up configuration from the config file.
            Dim factory As New ChannelFactory(Of ISampleServiceChannel)("WSHttpBinding_ISampleService")

            ' Add the client side behavior programmatically to all created channels.
            factory.Endpoint.Behaviors.Add(New EndpointBehaviorMessageInspector())

            Dim wcfClientChannel As ISampleServiceChannel = factory.CreateChannel()

            ' Making calls.
            Console.WriteLine("Enter the greeting to send: ")
            Dim greeting As String = Console.ReadLine()
            Console.WriteLine("The service responded: " & wcfClientChannel.SampleMethod(greeting))

            Console.WriteLine("Press ENTER to exit:")
            Console.ReadLine()

            ' Done with service.
            wcfClientChannel.Close()
            Console.WriteLine("Done!")
        Catch timeProblem As TimeoutException
            Console.WriteLine("The service operation timed out. " & timeProblem.Message)
            Console.Read()
        Catch fault As FaultException(Of SampleFault)
            Console.WriteLine("SampleFault fault occurred: {0}", fault.Detail.FaultMessage)
            Console.Read()
        Catch commProblem As CommunicationException
            Console.WriteLine("There was a communication problem. " & commProblem.Message)
            Console.Read()
        End Try
    End Sub

```

See Also

[<behaviors>](#)

Securing Clients

10/24/2018 • 7 minutes to read • [Edit Online](#)

In Windows Communication Foundation (WCF), the service dictates the security requirements for clients. That is, the service specifies what security mode to use, and whether or not the client must provide a credential. The process of securing a client, therefore, is simple: use the metadata obtained from the service (if it is published) and build a client. The metadata specifies how to configure the client. If the service requires that the client supply a credential, then you must obtain a credential that fits the requirement. This topic discusses the process in further detail. For more information about creating a secure service, see [Securing Services](#).

The Service Specifies Security

By default, WCF bindings have security features enabled. (The exception is the [BasicHttpBinding](#).) Therefore, if the service was created using WCF, there is a greater chance that it will implement security to ensure authentication, confidentiality, and integrity. In that case, the metadata the service provides will indicate what it requires to establish a secure communication channel. If the service metadata does not include any security requirements, there is no way to impose a security scheme, such as Secure Sockets Layer (SSL) over HTTP, on a service. If, however, the service requires the client to supply a credential, then the client developer, deployer, or administrator must supply the actual credential that the client will use to authenticate itself to the service.

Obtaining Metadata

When creating a client, the first step is to obtain the metadata for the service that the client will communicate with. This can be done in two ways. First, if the service publishes a metadata exchange (MEX) endpoint or makes its metadata available over HTTP or HTTPS, you can download the metadata using the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#), which generates both code files for a client as well as a configuration file. (For more information about using the tool, see [Accessing Services Using a WCF Client](#).) If the service does not publish a MEX endpoint and also does not make its metadata available over HTTP or HTTPS, you must contact the service creator for documentation that describes the security requirements and the metadata.

IMPORTANT

It is recommended that the metadata come from a trusted source and that it not be tampered with. Metadata retrieved using the HTTP protocol is sent in clear text and can be tampered with. If the service uses the [HttpsGetEnabled](#) and [HttpsGetUrl](#) properties, use the URL the service creator supplied to download the data using the HTTPS protocol.

Validating Security

Metadata sources can be divided into two broad categories: trust sources and untrusted sources. If you trust a source and have downloaded the client code and other metadata from that source's secure MEX endpoint, then you can build the client, supply it with the right credentials, and run it with no other concerns.

However, if you elect to download a client and metadata from a source that you know little about, be sure to validate the security measures the code uses. For example, you must not simply create a client that sends your personal or financial information to a service unless the service demands confidentiality and integrity (at the very least). You should trust the owner of the service to the extent that you are willing to disclose such information because such information will be visible to him or her.

As a rule, therefore, when using code and metadata from an untrusted source, check the code and metadata to ensure that it meets the security level that you require.

Setting a Client Credential

Setting a client credential on a client consists of two steps:

1. Determine the *client credential type* the service requires. This is accomplished by one of two methods. First, if you have documentation from the service creator, it should specify the client credential type (if any) the service requires. Second, if you have only a configuration file generated by the Svcutil.exe tool, you can examine the individual bindings to determine what credential type is required.
2. Specify an actual client credential. The actual client credential is called a *client credential value* to distinguish it from the type. For example, if the client credential type specifies a certificate, you must supply an X.509 certificate that is issued by a certification authority the service trusts.

Determining the Client Credential Type

If you have the configuration file the Svcutil.exe tool generated, examine the [bindings](#) section to determine what client credential type is required. Within the section are binding elements that specify the security requirements. Specifically, examine the <security> Element of each binding. That element includes the `mode` attribute, which you can set to one of three possible values (`Message` , `Transport` , or `TransportWithMessageCredential`). The value of the attribute determines the mode, and the mode determines which of the child elements is significant.

The <security> element can contain either a <transport> or <message> element, or both. The significant element is the one that matches the security mode. For example, the following code specifies that the security mode is `"Message"` , and the client credential type for the <message> element is `"Certificate"` . In this case, the <transport> element can be ignored. However, the <message> element specifies that an X.509 certificate must be supplied.

```
<wsHttpBinding>
  <binding name="WSHttpBinding_ICalculator">
    <security mode="Message">
      <transport clientCredentialType="Windows"
        realm="" />
      <message clientCredentialType="Certificate"
        negotiateServiceCredential="true"
        algorithmSuite="Default"
        establishSecurityContext="true" />
    </security>
  </binding>
</wsHttpBinding>
```

Note that if the `clientCredentialType` attribute is set to `"Windows"` , as shown in the following example, you do not need to supply an actual credential value. This is because the Windows integrated security provides the actual credential (a Kerberos token) of the person who is running the client.

```
<security mode="Message">
  <transport clientCredentialType="Windows "
    realm="" />
</security>
```

Setting the Client Credential Value

If it is determined that the client must supply a credential, use the appropriate method to configure the client. For example, to set a client certificate, use the [SetCertificate](#) method.

A common form of credential is the X.509 certificate. You can supply the credential in two ways:

- By programming it in your client code (using the `SetCertificate` method).

By adding a [<behaviors>](#) section of the configuration file for the client and using the `clientCredentials` element (shown below).

Setting a `<clientCredentials>` Value in Code

To set a `<clientCredentials>` value in code, you must access the `ClientCredentials` property of the `ClientBase<TChannel>` class. The property returns a `ClientCredentials` object that allows access to various credential types, as shown in the following table.

CLIENTCREDENTIAL PROPERTY	DESCRIPTION	NOTES
ClientCertificate	Returns an X509CertificateInitiatorClientCredential	Represents an X.509 certificate provided by the client to authenticate itself to the service.
HttpDigest	Returns an HttpDigestClientCredential	Represents an HTTP digest credential. The credential is a hash of the user name and password.
IssuedToken	Returns an IssuedTokenClientCredential	Represents a custom security token issued by a Security Token Service, commonly used in federation scenarios.
Peer	Returns a PeerCredential	Represents a Peer credential for participation in a Peer mesh on a Windows domain.
ServiceCertificate	Returns an X509CertificateRecipientClientCredential	Represents an X.509 certificate provided by the service in an out-of-band negotiation.
UserName	Returns a UserNamePasswordClientCredential	Represents a user name and password pair.
Windows	Returns a WindowsClientCredential	Represents a Windows client credential (a Kerberos credential). The properties of the class are read-only.

Setting a `<clientCredentials>` Value in Configuration

Credential values are specified by using an endpoint behavior as child elements of the `<clientCredentials>` element. The element used depends on the client credential type. For example, the following example shows the configuration to set an X.509 certificate using the `<<clientCertificate>`.

```
<configuration>
  <system.serviceModel>
    <behaviors>
      <endpointBehaviors>
        <behavior name="myEndpointBehavior">
          <clientCredentials>
            <clientCertificate findvalue="myMachineName"
              storeLocation="Current" X509FindType="FindBySubjectName" />
          </clientCredentials>
        </behavior>
      </endpointBehaviors>
    </system.serviceModel>
  </configuration>
```

To set the client credential in configuration, add an `<endpointBehaviors>` element to the configuration file. Additionally, the added behavior element must be linked to the service's endpoint using the

`behaviorConfiguration` attribute of the [<endpoint>](#) element as shown in the following example. The value of the `behaviorConfiguration` attribute must match the value of the `name` attribute.

```
<configuration>

<system.serviceModel>

<client>

<endpoint address="http://localhost/servicemodelsamples/service.svc"

binding="wsHttpBinding"

bindingConfiguration="Binding1"

behaviorConfiguration="myEndpointBehavior"

contract="Microsoft.ServiceModel.Samples.ICalculator" />

</client>

</system.serviceModel>

</configuration>
```

NOTE

Some of the client credential values cannot be set using application configuration files, for example, user name and password, or Windows user and password values. Such credential values can be specified only in code.

For more information about setting the client credential, see [How to: Specify Client Credential Values](#).

NOTE

`ClientCredentialType` is ignored when `SecurityMode` is set to `"TransportWithMessageCredential"`, as shown in the following sample configuration.

```
<wsHttpBinding>
  <binding name="PingBinding">
    <security mode="TransportWithMessageCredential" >
      <message clientCredentialType="UserName"
        establishSecurityContext="false"
        negotiateServiceCredential="false" />
      <transport clientCredentialType="Certificate" />
    </security>
  </binding>
</wsHttpBinding>
```

See Also

[ClientCredentials](#)

[ClientBase<TChannel>](#)

[ClientCredentials](#)

[HttpsGetEnabled](#)

[HttpsGetUrl](#)

[<bindings>](#)

[Configuration Editor Tool \(SvcConfigEditor.exe\)](#)

[Securing Services](#)

[Accessing Services Using a WCF Client](#)

[How to: Specify Client Credential Values](#)

[ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#)

[How to: Specify the Client Credential Type](#)

How to: Specify Client Credential Values

5/5/2018 • 3 minutes to read • [Edit Online](#)

Using Windows Communication Foundation (WCF), the service can specify how a client is authenticated to the service. For example, a service can stipulate that the client be authenticated with a certificate.

To determine the client credential type

1. Retrieve metadata from the service's metadata endpoint. The metadata typically consists of two files: the client code in the programming language of your choice (the default is Visual C#), and an XML configuration file. One way to retrieve metadata is to use the Svcutil.exe tool to return the client code and client configuration. For more information, see [Retrieving Metadata](#) and [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#).
2. Open the XML configuration file. If you use the Svcutil.exe tool, the default name of the file is Output.config.
3. Find the **<security>** element with the **mode** attribute (**<security mode = MessageOrTransport >** where **MessageOrTransport** is set to one of the security modes).
4. Find the child element that matches the mode value. For example, if the mode is set to **Message**, find the **<message>** element contained in the **<security>** element.
5. Note the value assigned to the **clientCredentialType** attribute. The actual value depends on which mode is used, transport or message.

The following XML code shows configuration for a client using message security and requiring a certificate to authenticate the client.

```
<security mode="Message">
  <transport clientCredentialType="Windows" proxyCredentialType="None"
    realm="" />
  <message clientCredentialType="Certificate" negotiateServiceCredential="true"
    algorithmSuite="Default" establishSecurityContext="true" />
</security>
```

Example: TCP Transport Mode with Certificate as Client Credential

This example sets the security mode to Transport mode and sets the client credential value to an X.509 certificate. The following procedures demonstrate how to set the client credential value on the client in code and configuration. This assumes that you have used the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) to return the metadata (code and configuration) from the service. For more information, see [How to: Create a Client](#).

To specify the client credential value on the client in code

1. Use the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) to generate code and configuration from the service.
2. Create an instance of the WCF client using the generated code.
3. On the client class, set the [ClientCredentials](#) property of the [ClientBase<TChannel>](#) class to an appropriate value. This example sets the property to an X.509 certificate using the [SetCertificate](#) method of the [X509CertificateInitiatorClientCredential](#) class.

```

// Create a binding using Transport and a certificate.
NetTcpBinding b = new NetTcpBinding();
b.Security.Mode = SecurityMode.Transport;
b.Security.Transport.ClientCredentialType =
    TcpClientCredentialType.Certificate;

// Create an EndPointAddress.
EndpointAddress ea = new EndpointAddress(
    "net.tcp://localhost:8036/Calculator/MyCalculator");

// Create the client.
CalculatorClient cc = new CalculatorClient(b, ea);

// Set the certificate for the client.
cc.ClientCredentials.ClientCertificate.SetCertificate(
    StoreLocation.LocalMachine,
    StoreName.My,
    X509FindType.FindBySubjectName,
    "cohowinery.com");
try
{
    cc.Open();
    // Begin using the client.
    Console.WriteLine(cc.Divide(1001, 2));
    cc.Close();
}
catch (AddressAccessDeniedException adExc)
{
    Console.WriteLine(adExc.Message);
    Console.ReadLine();
}
catch (System.Exception exc)
{
    Console.WriteLine(exc.Message);
    Console.ReadLine();
}
}

```

```

' Create a binding using Transport and a certificate.
Dim b As New NetTcpBinding()
b.Security.Mode = SecurityMode.Transport
b.Security.Transport.ClientCredentialType = TcpClientCredentialType.Certificate

' Create an EndPointAddress.
Dim ea As New EndpointAddress("net.tcp://localhost:8036/Calculator/MyCalculator")

' Create the client.
Dim cc As New CalculatorClient(b, ea)

' Set the certificate for the client.
cc.ClientCredentials.ClientCertificate.SetCertificate( _
    StoreLocation.LocalMachine, StoreName.My, X509FindType.FindBySubjectName, "cohowinery.com")
Try
    cc.Open()
    ' Begin using the client.
    Console.WriteLine(cc.Divide(1001, 2))
    cc.Close()
Catch adExc As AddressAccessDeniedException
    Console.WriteLine(adExc.Message)
    Console.ReadLine()
Catch exc As System.Exception
    Console.WriteLine(exc.Message)
    Console.ReadLine()
End Try

```

You can use any of the enumerations of the [X509FindType](#) class. The subject name is used here in case the

certificate is changed (due to an expiration date). Using the subject name enables the infrastructure to find the certificate again.

To specify the client credential value on the client in configuration

1. Add a `<behavior>` element to the `<behaviors>` element.
2. Add a `<clientCredentials>` element to the `<behaviors>` element. Be sure to set the required `name` attribute to an appropriate value.
3. Add a `<clientCertificate>` element to the `<clientCredentials>` element.
4. Set the following attributes to appropriate values: `storeLocation`, `storeName`, `x509FindType`, and `findValue`, as shown in the following code. For more information about certificates, see [Working with Certificates](#).

```
<behaviors>
  <endpointBehaviors>
    <behavior name="endpointCredentialBehavior">
      <clientCredentials>
        <clientCertificate findValue="Contoso.com"
                           storeLocation="LocalMachine"
                           storeName="TrustedPeople"
                           x509FindType="FindBySubjectName" />
      </clientCredentials>
    </behavior>
  </endpointBehaviors>
</behaviors>
```

5. When configuring the client, specify the behavior by setting the `behaviorConfiguration` attribute of the `<endpoint>` element, as shown in the following code. The endpoint element is a child of the `<client>` element. Also, specify the name of the binding configuration by setting the `bindingConfiguration` attribute to the binding for the client. If you are using a generated configuration file, the binding's name is automatically generated. In this example, the name is `"tcpBindingWithCredential"`.

```
<client>
  <endpoint name=""
            address="net.tcp://contoso.com:8036/aloha"
            binding="netTcpBinding"
            bindingConfiguration="tcpBindingWithCredential"
            behaviorConfiguration="endpointCredentialBehavior" />
</client>
```

See Also

[NetTcpBinding](#)

[SetCertificate](#)

[X509CertificateRecipientServiceCredential](#)

[ClientBase<TChannel>](#)

[X509CertificateInitiatorClientCredential](#)

[Programming WCF Security](#)

[Selecting a Credential Type](#)

[ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#)

[Working with Certificates](#)

[How to: Create a Client](#)

[<netTcpBinding>](#)

[<security>](#)

<message>
<behavior>
<behaviors>
<clientCertificate>
<clientCredentials>

Introduction to Extensibility

5/5/2018 • 2 minutes to read • [Edit Online](#)

The Windows Communication Foundation (WCF) application model is designed to solve the greater part of the communication requirements of any distributed application. But there are always scenarios that the default application model and system-provided implementations do not support. The WCF extensibility model is intended to support custom scenarios by enabling you to modify system behavior at every level, even to the point of replacing the entire application model. This topic outlines the various areas of extension and points to more information about each.

Areas to Extend

You can extend:

- The application runtime. This extends the dispatching and the processing of messages for the application. This area also includes extending the security system, the metadata system, the serialization system, and the bindings and binding elements that connect the application with the underlying channel system.
- The channel and channel runtime. This extends the system that functions at the message level, providing protocol, transport, and encoding support.
- The host runtime. This extends the relationship of the hosting application domain to the channel and application runtime.

Extending the Application Runtime

In WCF applications, there is a distinction between messages that are destined for a corresponding channel and messages that are destined for the application itself. Channel messages support some channel-related functionality, such as establishing a secure conversation or establishing a reliable session. These messages are not available to the application runtime; they are processed before the application layer is involved.

Application messages contain data that is destined for a client or service operation that you or your customer has created. These messages are available to the application-level extension system in message or object form, depending upon your needs.

All messages pass through the channel system; only application messages are passed from the channel system into the application. To create new channel-level functionality, you must extend the channel system. To create new application-level functionality, you must extend the service or client runtime (dispatchers and channel factories, respectively). For more information about extending the application runtime, see [Extending ServiceHost and the Service Model Layer](#).

Extending Security

To build custom security mechanisms such as tokens and credentials, you must extend the security system. For more information, see [Extending Security](#).

Extending Metadata

To expose your metadata in differently than the default, you must extend the metadata system. For more information, see [Extending the Metadata System](#).

Extending Serialization

To build custom encoders, provide data surrogates, or other work involving the customization of transferred data, you must extend the serialization system. For more information, see [Extending Encoders and Serializers](#).

Extending Bindings

To associate transport or protocol channels with the application layer, you must extend the binding system. For more information, see [Extending Bindings](#).

Extending the Channel System

To create channels that support custom transports or protocol functionality, see [Extending the Channel Layer](#).

Extending the Service Hosting System

To modify the service-wide application model, you must extend [System.ServiceModel.ServiceHostBase](#) class. For more information, see [Extending ServiceHost and the Service Model Layer](#).

To modify the relationship between the hosting application domain and the service host, you must extend the [System.ServiceModel.Activation.ServiceHostFactory](#) class. For more information, see [Extending Hosting Using ServiceHostFactory](#).

See Also

[Extending WCF](#)

WCF Troubleshooting Quickstart

8/31/2018 • 11 minutes to read • [Edit Online](#)

This topic lists a number of known issues customers have run into while developing WCF clients and services. If the issue you are running into is not in this list, we recommend you configure tracing for your service. This will generate a trace file that you can view with the trace file viewer and get detailed information about exceptions that may be occurring within the service. For more information on configuring tracing, see: [Configuring Tracing](#). For more information on the trace file viewer, see: [Service Trace Viewer Tool \(SvcTraceViewer.exe\)](#).

1. [After installing Windows 7 and IIS, when I attempt to browse to a WCF service I get the following error message: HTTP Error 404.3 – Not Found](#)

HTTP Error 404.3 – Not FoundThe page you are requesting cannot be served because of the extension configuration. If the page is a script, add a handler. If the file should be downloaded, add a MIME map. Detailed Error InformationModule StaticFileModule.

2. [Sometimes I receive a MessageSecurityException on the second request if my client is idle for a while after the first request. What is happening?](#)
3. [My service starts to reject new clients after about 10 clients are interacting with it. What is happening?](#)
4. [Can I load my service configuration from somewhere other than the WCF application's configuration file?](#)
5. [My service and client work great, but I can't get them to work when the client is on another computer? What's happening?](#)
6. [When I throw a FaultException<Exception> where the type is an exception, I always receive a general FaultException type on the client and not the generic type. What's happening?](#)
7. [It seems like one-way and request-reply operations return at roughly the same speed when the reply contains no data. What's happening?](#)
8. [I'm using an X.509 certificate with my service and I get a System.Security.Cryptography.CryptographicException. What's happening?](#)
9. [I changed the first parameter of an operation from uppercase to lowercase; now my client throws an exception. What's happening?](#)
10. [I'm using one of my tracing tools and I get an EndpointNotFoundException. What's happening?](#)
11. [When calling a WCF Web HTTP application from a WCF SOAP application the service returns the following error: 405 Method Not Allowed](#)

[What is the base address? How does it relate to an endpoint address?](#)

After installing Windows 7 and IIS, when I attempt to browse to a WCF service I get the following error message: HTTP Error 404.3 – Not Found

The full error message is:

HTTP Error 404.3 – Not FoundThe page you are requesting cannot be served because of the extension configuration. If the page is a script, add a handler. If the file should be downloaded, add a MIME map. Detailed Error InformationModule StaticFileModule.

This error message occurs when "Windows Communication Foundation HTTP Activation" is not explicitly set in the Control Panel. To set this go to the Control Panel, click Programs in the lower left hand corner of the window. Click Turn Windows features on or off. Expand Microsoft .NET Framework 3.5.1 and select Windows Communication Foundation HTTP Activation.

Sometimes I receive a `MessageSecurityException` on the second request if my client is idle for a while after the first request. What is happening?

The second request can fail primarily for two reasons: (1) the session has timed out or (2) the Web server that is hosting the service is recycled. In the first case, the session is valid until the service times out. When the service does not receive a request from the client within the period of time specified in the service's binding ([ReceiveTimeout](#)), the service terminates the security session. Subsequent client messages result in the `MessageSecurityException`. The client must re-establish a secure session with the service to send future messages or use a stateful security context token. Stateful security context tokens also allow a secure session to survive a Web server being recycled. For more information about using stateful secure context tokens in a secure session, see [How to: Create a Security Context Token for a Secure Session](#). Alternatively, you can disable secure sessions. When you use the `<wsHttpBinding>` binding, you can set the `establishSecurityContext` property to `false` to disable secure sessions. To disable secure sessions for other bindings, you must create a custom binding. For details about creating a custom binding, see [How to: Create a Custom Binding Using the SecurityBindingElement](#). Before you apply any of these options, you must understand your application's security requirements.

My service starts to reject new clients after about 10 clients are interacting with it. What is happening?

By default, services can have only 10 concurrent sessions. Therefore, if the service bindings use sessions, the service accepts new client connections until it reaches that number, after which it refuses new client connections until one of the current sessions ends. You can support more clients in a number of ways. If your service does not require sessions, do not use a sessionful binding. (For more information, see [Using Sessions](#).) Another option is to increase the session limit by changing the value of the `MaxConcurrentSessions` property to the number appropriate to your circumstance.

Can I load my service configuration from somewhere other than the WCF application's configuration file?

Yes, however, you have to create a custom `ServiceHost` class that overrides the `ApplyConfiguration` method. Inside that method, you can call the base to load configuration first (if you want to load the standard configuration information as well) but you can also entirely replace the configuration loading system. Note that if you want to load configuration from a configuration file that is different from the application configuration file, you must parse the configuration file yourself and load the configuration.

The following code example shows how to override the `ApplyConfiguration` method and directly configure an endpoint.

```

public class MyServiceHost : ServiceHost
{
    public MyServiceHost(Type serviceType, params Uri[] baseAddresses)
        : base(serviceType, baseAddresses)
    {
        Console.WriteLine("MyServiceHost Constructor");
    }

    protected override void ApplyConfiguration()
    {
        string straddress = GetAddress();
        Uri address = new Uri(straddress);
        Binding binding = GetBinding();
        base.AddServiceEndpoint(typeof(IData), binding, address);
    }

    string GetAddress()
    {
        return "http://MyMachine:7777/MyEndpointAddress/";
    }

    Binding GetBinding()
    {
        WSHttpBinding binding = new WSHttpBinding();
        binding.Security.Mode = SecurityMode.None;
        return binding;
    }
}

```

My service and client work great, but I can't get them to work when the client is on another computer? What's happening?

Depending upon the exception, there may be several issues:

- You might need to change the client endpoint addresses to the host name and not "localhost".
- You might need to open the port to the application. For details, see [Firewall Instructions](#) from the SDK samples.
- For other possible issues, see the samples topic [Running the Samples in a Workgroup and Across Machines](#).
- If your client is using Windows credentials and the exception is a [SecurityNegotiationException](#), configure Kerberos as follows.

1. Add the identity credentials to the endpoint element in the client's App.config file:

```

<endpoint
    address="http://MyServer:8000/MyService/"
    binding="wsHttpBinding"
    bindingConfiguration="WSHttpBinding_IServiceExample"
    contract="IServiceExample"
    behaviorConfiguration="ClientCredBehavior"
    name="WSHttpBinding_IServiceExample">
    <identity>
        <userPrincipalName value="name@corp.contoso.com"/>
    </identity>
</endpoint>

```

2. Run the self-hosted service under the System or NetworkService account. You can run this command to create a command window under the System account:

```
at 12:36 /interactive "cmd.exe"
```

3. Host the service under Internet Information Services (IIS), which, by default, uses the service principal name (SPN) account.
4. Register a new SPN with the domain using SetSPN. Note that you will need to be a domain administrator in order to do this.

For more information about the Kerberos protocol, see [Security Concepts Used in WCF](#) and:

- [Debugging Windows Authentication Errors](#)
- [Registering Kerberos Service Principal Names by Using Http.sys](#)
- [Kerberos Explained](#)

When I throw a `FaultException<Exception>` where the type is an exception, I always receive a general `FaultException` type on the client and not the generic type. What's happening?

It is highly recommended that you create your own custom error data type and declare that as the detail type in your fault contract. The reason is that using system-provided exception types:

- Creates a type dependency that removes one of the biggest strengths of service-oriented applications.
- Cannot depend upon exceptions serializing in a standard way. Some—like [SecurityException](#)—may not be serializable at all.
- Exposes internal implementation details to clients. For more information, see [Specifying and Handling Faults in Contracts and Services](#).

If you are debugging an application, however, you can serialize exception information and return it to the client by using the [ServiceDebugBehavior](#) class.

It seems like one-way and request-reply operations return at roughly the same speed when the reply contains no data. What's happening?

Specifying that an operation is one way means only that the operation contract accepts an input message and does not return an output message. In WCF, all client invocations return when the outbound data has been written to the wire or an exception is thrown. One-way operations work the same way, and they can throw if the service cannot be located or block if the service is not prepared to accept the data from the network. Typically in WCF, this results in one-way calls returning to the client more quickly than request-reply; but any condition that slows the sending of the outbound data over the network slows one-way operations as well as request-reply operations. For more information, see [One-Way Services](#) and [Accessing Services Using a WCF Client](#).

I'm using an X.509 certificate with my service and I get a `System.Security.Cryptography.CryptographicException`. What's happening?

This commonly occurs after changing the user account under which the IIS worker process runs. For example, in Windows XP, if you change the default user account that the `Aspnet_wp.exe` runs under from ASPNET to a custom user account, you may see this error. If using a private key, the process that uses it will need to have permissions to access the file storing that key.

If this is the case, you must give read access privileges to the process's account for the file containing the private key. For example, if the IIS worker process is running under the Bob account, then you will need to give Bob read access to the file containing the private key.

For more information about how to give the correct user account access to the file that contains the private key for a specific X.509 certificate, see [How to: Make X.509 Certificates Accessible to WCF](#).

I changed the first parameter of an operation from uppercase to lowercase; now my client throws an exception. What's happening?

The value of the parameter names in the operation signature are part of the contract and are case-sensitive. Use the [System.ServiceModel.MessageParameterAttribute](#) attribute when you need to distinguish between the local parameter name and the metadata that describes the operation for client applications.

I'm using one of my tracing tools and I get an EndpointNotFoundException. What's happening?

If you are using a tracing tool that is not the system-provided WCF tracing mechanism and you receive an [EndpointNotFoundException](#) that indicates that there was an address filter mismatch, you need to use the [ClientViaBehavior](#) class to direct the messages to the tracing utility and have the utility redirect those messages to the service address. The [ClientViaBehavior](#) class alters the `Via` addressing header to specify the next network address separately from the ultimate receiver, indicated by the `To` addressing header. When doing this, however, do not change the endpoint address, which is used to establish the `To` value.

The following code example shows an example client configuration file.

```
<endpoint
  address=http://localhost:8000/MyServer/
  binding="wsHttpBinding"
  bindingConfiguration="WSHttpBinding_IMyContract"
  behaviorConfiguration="MyClient"
  contract="IMyContract"
  name="WSHttpBinding_IMyContract">
</endpoint>
<behaviors>
  <endpointBehaviors>
    <behavior name="MyClient">
      <clientVia viaUri="http://localhost:8001/MyServer/" />
    </behavior>
  </endpointBehaviors>
</behaviors>
```

What is the base address? How does it relate to an endpoint address?

A base address is the root address for a [ServiceHost](#) class. By default, if you add a [ServiceMetadataBehavior](#) class into your service configuration, the Web Services Description Language (WSDL) for all endpoints the host publishes are retrieved from the HTTP base address, plus any relative address provided to the metadata behavior, plus "?wsdl". If you are familiar with ASP.NET and IIS, the base address is equivalent to the virtual directory.

Sharing a port between a service endpoint and a mex endpoint using the NetTcpBinding

If you specify the base address for a service as `net.tcp://MyServer:8080/MyService` and add the following endpoints:

```
<services>
  <service name="Microsoft.Samples.NetTcp.CalculatorService">
    <endpoint address="calcsvc" binding="netTcpBinding" contract="Microsoft.Samples.NetTcp.ICalculator"/>
    <endpoint address="mex" binding="mexTcpBinding" contract="IMetadataExchange" />
  </service>
</services>
```

And if you modify one of the NetTcpBinding settings as shown in the following configuration snippet:

```
<bindings>
  <netTcpBinding>
    <binding closeTimeout="00:01:00" openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
transactionFlow="false" transferMode="Buffered" transactionProtocol="OleTransactions"
hostNameComparisonMode="StrongWildcard" listenBacklog="10" maxBufferPoolSize="524288" maxBufferSize="65536"
maxConnections="11" maxReceivedMessageSize="65536">
      <readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384" maxBytesPerRead="4096"
maxNameTableCharCount="16384"/>
      <reliableSession ordered="true" inactivityTimeout="00:10:00" enabled="false"/>
      <security mode="Transport">
        <transport clientCredentialType="Windows" protectionLevel="EncryptAndSign"/>
      </security>
    </binding>
  </netTcpBinding>
</bindings>
```

You will see an error like the following: Unhandled Exception:

System.ServiceModel.AddressAlreadyInUseException: There is already a listener on IP endpoint 0.0.0.0:9000 You can work around this error by specifying a fully qualified URL with a different port for the MEX endpoint as shown in the following configuration snippet:

```
<services>
  <service name="Microsoft.Samples.NetTcp.CalculatorService">
    <endpoint address="calcsvc" binding="netTcpBinding" contract="Microsoft.Samples.NetTcp.ICalculator"/>
    <endpoint address="net.tcp://localhost:9001/servicemodelsamples/mex" binding="mexTcpBinding"
contract="IMetadataExchange" />
  </service>
</services>
```

When calling a WCF Web HTTP application from a WCF SOAP application the service returns the following error: 405 Method Not Allowed

Calling a WCF Web HTTP application (a service that uses the [WebHttpBinding](#) and [WebHttpBehavior](#)) from a WCF service may generate the following exception: `Unhandled Exception: System.ServiceModel.FaultException`1[System.ServiceModel.ExceptionDetail]: The remote server returned an unexpected response: (405) Method Not Allowed.` This exception occurs because WCF overwrites the outgoing [OperationContext](#) with the incoming [OperationContext](#). To solve this problem create an [OperationContextScope](#) within the WCF Web HTTP service operation. For example:

```
public string Echo(string input)
{
    using (new OperationContextScope(this.InnerChannel))
    {
        return base.Channel.Echo(input);
    }
}
```

See Also

[Debugging Windows Authentication Errors](#)

WCF Error Handling

9/19/2018 • 2 minutes to read • [Edit Online](#)

The errors encountered by a WCF application belong to one of three groups:

1. Communication Errors
2. Proxy/Channel Errors
3. Application Errors

Communication errors occur when a network is unavailable, a client uses an incorrect address, or the service host is not listening for incoming messages. Errors of this type are returned to the client as [CommunicationException](#) or [CommunicationException](#)-derived classes.

Proxy/Channel errors are errors that occur within the channel or proxy itself. Errors of this type include: attempting to use a proxy or channel that has been closed, a contract mismatch exists between the client and service, or the client's credentials are rejected by the service. There are many different types of errors in this category, too many to list here. Errors of this type are returned to the client as-is (no transformation is performed on the exception objects).

Application errors occur during the execution of a service operation. Errors of this kind are sent to the client as [FaultException](#) or [FaultException<TDetail>](#).

Error handling in WCF is performed by one or more of the following:

- Directly handling the exception thrown. This is only done for communication and proxy/channel errors.
- Using fault contracts
- Implementing the [IErrorHandler](#) interface
- Handling [ServiceHost](#) events

Fault Contracts

Fault contracts allow you to define the errors that can occur during service operation in a platform independent way. By default all exceptions thrown from within a service operation will be returned to the client as a [FaultException](#) object. The [FaultException](#) object will contain very little information. You can control the information sent to the client by defining a fault contract and returning the error as a [FaultException<TDetail>](#). For more information, see [Specifying and Handling Faults in Contracts and Services](#).

IErrorHandler

The [IErrorHandler](#) interface allows you more control over how your WCF application responds to errors. It gives you full control over the fault message that is returned to the client and allows you to perform custom error processing such as logging. For more information about [IErrorHandler](#) and [Extending Control Over Error Handling and Reporting](#)

ServiceHost Events

The [ServiceHost](#) class hosts services and defines several events that may be needed for handling errors. For example:

1. [Faulted](#)
2. [UnknownMessageReceived](#)

For more information, see [ServiceHost](#)

WCF and ASP.NET Web API

10/13/2018 • 2 minutes to read • [Edit Online](#)

WCF is Microsoft's unified programming model for building service-oriented applications. It enables developers to build secure, reliable, transacted solutions that integrate across platforms and interoperate with existing investments. [ASP.NET Web API](#) is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices. ASP.NET Web API is an ideal platform for building RESTful applications on the .NET Framework. This topic presents some guidance to help you decide which technology will best meet your needs.

Choosing which technology to use

The following table describes the major features of each technology.

WCF	ASP.NET WEB API
Enables building services that support multiple transport protocols (HTTP, TCP, UDP, and custom transports) and allows switching between them.	HTTP only. First-class programming model for HTTP. More suitable for access from various browsers, mobile devices etc enabling wide reach.
Enables building services that support multiple encodings (Text, MTOM, and Binary) of the same message type and allows switching between them.	Enables building Web APIs that support wide variety of media types including XML, JSON etc.
Supports building services with WS-* standards like Reliable Messaging, Transactions, Message Security.	Uses basic protocol and formats such as HTTP, WebSockets, SSL, JSON, and XML. There is no support for higher level protocols such as Reliable Messaging or Transactions.
Supports Request-Reply, One Way, and Duplex message exchange patterns.	HTTP is request/response but additional patterns can be supported through SignalR and WebSockets integration.
WCF SOAP services can be described in WSDL allowing automated tools to generate client proxies even for services with complex schemas.	There is a variety of ways to describe a Web API ranging from auto-generated HTML help page describing snippets to structured metadata for OData integrated APIs.
Ships with the .NET framework.	Ships with .NET framework but is open-source and is also available out-of-band as independent download.

Use WCF to create reliable, secure web services that are accessible over a variety of transports. Use ASP.NET Web API to create HTTP-based services that are accessible from a wide variety of clients. Use ASP.NET Web API if you are creating and designing new REST-style services. Although WCF provides some support for writing REST-style services, the support for REST in ASP.NET Web API is more complete and all future REST feature improvements will be made in ASP.NET Web API. If you have an existing WCF service and you want to expose additional REST endpoints, use WCF and the [WebHttpBinding](#).

See Also

[What Is Windows Communication Foundation](#)

[Fundamental Windows Communication Foundation Concepts](#)

WCF Feature Details

5/4/2018 • 2 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) allows extensive control over the messaging functions of an application. The topics in this section go into detail about the available features. For more information about basic programming, see [Basic WCF Programming](#).

In This Section

[Workflow Services](#)

Describes how to create and configure workflow services.

[Endpoints: Addresses, Bindings, and Contracts](#)

Describes how to control multiple aspects of your service.

[Data Transfer and Serialization](#)

Describes how serialization of data can be tailored for interoperability or future compatibility.

[Sessions, Instancing, and Concurrency](#)

Describes the instancing and session modes of WCF and how to select the right mode for your application.

[Transports](#)

Describes how to configure the transport layer, the lowest level of the channel stack.

[Queues and Reliable Sessions](#)

Describes queues, which store messages from a sending application on behalf of a receiving application and later forward these messages to the receiving application.

[Transactions](#)

Explains how to create transacted operations that can be rolled back if needed.

[Security](#)

Describes how WCF security helps you to create applications that have confidentiality and integrity. Authentication and authorization are also available, as are auditing features.

[Peer-to-Peer Networking](#)

Details how to create peer services and clients.

[Metadata](#)

Describes metadata architecture and formats.

[Clients](#)

Describes how to create a variety of clients that access services.

[Hosting](#)

Describes hosting. A service can be hosted by another application, or it can be self-hosted.

[Interoperability and Integration](#)

Describes how to use WCF to extend your existing logic rather than having to rewrite it if you have a substantial investment in component-based application logic hosted in COM+.

[WCF Web HTTP Programming Model](#)

Describes the WCF Web Programming Model that allows developers to expose WCF service operations to non-SOAP endpoints.

[WCF Syndication](#)

Describes support to easily expose syndication feeds from a WCF service.

[AJAX Integration and JSON Support](#)

Describes support for ASP.NET Asynchronous JavaScript and XML (AJAX) and the JavaScript Object Notation (JSON) data format to allow WCF services to expose operations to AJAX clients.

[WCF Discovery](#)

Describes support to enable services to be discoverable at runtime in an interoperable way using the WS-Discovery protocol.

[Routing](#)

Describes the routing service.

Reference

[System.ServiceModel](#)

[System.ServiceModel.Channels](#)

[System.IdentityModel.Selectors](#)

[System.ServiceModel.Routing](#)

Related Sections

[Basic WCF Programming](#)

Extending WCF

5/5/2018 • 2 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) allows you to modify and extend run time components to precisely control and extend service-based applications. The topics in this section go in depth about the extensibility architecture. For more information about basic programming, see [Basic WCF Programming](#).

In This Section

[Extending ServiceHost and the Service Model Layer](#)

The service model layer is responsible for pulling incoming messages out of the underlying channels, translating them into method invocations in application code, and sending the results back to the caller. Service model extensions modify or implement execution or communication behavior and features involving dispatcher functionality, custom behaviors, message and parameter interception, and other extensibility functionality.

[Extending Bindings](#)

Bindings are objects that describe the communication details required to connect to an endpoint. Binding extensions or custom bindings implement custom communication functionality required to support application features.

[Extending the Channel Layer](#)

The channel layer sits beneath the service model layer and is responsible for the exchange of messages between clients and services. Channel extensions can implement new protocol functionality, such as security. Channel extensions also transport functionality, such as implementing a new network transport to carry SOAP messages.

[Extending Security](#)

Security in WCF consists of transfer security (integrity, confidentiality, and authentication), access control (authorization) and auditing. The classes found in the `IdentityModel` namespace are used by WCF for access control. Understanding the security architecture allows you to create custom claim types to accommodate custom access control systems.

[Extending the Metadata System](#)

The WCF metadata system is a group of classes and interfaces that represent metadata required to implement service-based applications. Modify or extend the classes or implement and configure the interfaces to export and import custom metadata such as Web Services Description Language (WSDL) extensions or custom WS-PolicyAttachments assertions.

[Extending Encoders and Serializers](#)

Encoders and serializers translate data from one form to another. The topics in this section discuss how to extend the supplied classes to meet special requirements.

Reference

[System.ServiceModel](#)

[System.ServiceModel.Channels](#)

[System.ServiceModel.Description](#)

[System.IdentityModel.Claims](#)

[System.IdentityModel.Policy](#)

[System.IdentityModel.Selectors](#)

[System.IdentityModel.Tokens](#)

Related Sections

[Basic WCF Programming](#)

[WCF Feature Details](#)

[Guidelines and Best Practices](#)

Guidelines and Best Practices

8/31/2018 • 2 minutes to read • [Edit Online](#)

This section contains topics that provide guidelines for creating Windows Communication Foundation (WCF) applications.

In This Section

[Best Practices: Data Contract Versioning](#)

Explains how and when to create data contracts that do not break when future versions are created.

[Service Versioning](#)

Explains how to consider versioning in WCF. After deployment, services (and the endpoints they expose) might need to be changed, for example, to satisfy changing business requirements or IT requirements, or to fix issues. Each change introduces a new version of the service.

[Load Balancing](#)

Lists guidelines for load balancing with a Web farm.

[Controlling Resource Consumption and Improving Performance](#)

Describes the properties that are designed to help prevent undue resource consumption and improve security and points to more complete information about their use.

[Deploying WCF Applications with ClickOnce](#)

Describes the considerations to be made when using the ClickOnce feature.

Reference

[System.ServiceModel](#)

[System.Runtime.Serialization](#)

Related Sections

[Conceptual Overview](#)

[Basic WCF Programming](#)

See Also

[What Is Windows Communication Foundation](#)

[Windows Communication Foundation Samples](#)

[Conceptual Overview](#)

[Building Clients](#)

Best Practices: Data Contract Versioning

5/5/2018 • 9 minutes to read • [Edit Online](#)

This topic lists the best practices for creating data contracts that can evolve easily over time. For more information about data contracts, see the topics in [Using Data Contracts](#).

Note on Schema Validation

In discussing data contract versioning, it is important to note that the data contract schema exported by Windows Communication Foundation (WCF) does not have any versioning support, other than the fact that elements are marked as optional by default.

This means that even the most common versioning scenario, such as adding a new data member, cannot be implemented in a way that is seamless with regard to a given schema. The newer versions of a data contract (with a new data member, for example) do not validate using the old schema.

However, there are many scenarios in which strict schema compliance is not required. Many Web services platforms, including WCF and XML Web services created using ASP.NET, do not perform schema validation by default and therefore tolerate extra elements that are not described by the schema. When working with such platforms, many versioning scenarios are easier to implement.

Thus, there are two sets of data contract versioning guidelines: one set for scenarios where strict schema validity is important, and another set for scenarios when it is not.

Versioning When Schema Validation Is Required

If strict schema validity is required in all directions (new-to-old and old-to-new), data contracts should be considered immutable. If versioning is required, a new data contract should be created, with a different name or namespace, and the service contract using the data type should be versioned accordingly.

For example, a purchase order processing service contract named `PoProcessing` with a `PostPurchaseOrder` operation takes a parameter that conforms to a `PurchaseOrder` data contract. If the `PurchaseOrder` contract has to change, you must create a new data contract, that is, `PurchaseOrder2`, which includes the changes. You must then handle the versioning at the service contract level. For example, by creating a `PostPurchaseOrder2` operation that takes the `PurchaseOrder2` parameter, or by creating a `PoProcessing2` service contract where the `PostPurchaseOrder` operation takes a `PurchaseOrder2` data contract.

Note that changes in data contracts that are referenced by other data contracts also extend to the service model layer. For example, in the previous scenario the `PurchaseOrder` data contract does not need to change. However, it contains a data member of a `Customer` data contract, which in turn contained a data member of the `Address` data contract, which does need to be changed. In that case, you would need to create an `Address2` data contract with the required changes, a `Customer2` data contract that contains the `Address2` data member, and a `PurchaseOrder2` data contract that contains a `Customer2` data member. As in the previous case, the service contract would have to be versioned as well.

Although in these examples names are changed (by appending a "2"), the recommendation is to change namespaces instead of names by appending new namespaces with a version number or a date. For example, the

`http://schemas.contoso.com/2005/05/21/PurchaseOrder` data contract would change to the

`http://schemas.contoso.com/2005/10/14/PurchaseOrder` data contract.

For more information, see Best Practices: [Service Versioning](#).

Occasionally, you must guarantee strict schema compliance for messages sent by your application, but cannot rely on the incoming messages to be strictly schema-compliant. In this case, there is a danger that an incoming message might contain extraneous data. The extraneous values are stored and returned by WCF and thus results in schema-invalid messages being sent. To avoid this problem, the round-tripping feature should be turned off. There are two ways to do this.

- Do not implement the [IExtensibleDataObject](#) interface on any of your types.
- Apply a [ServiceBehaviorAttribute](#) attribute to your service contract with the [IgnoreExtensionDataObject](#) property set to `true`.

For more information about round-tripping, see [Forward-Compatible Data Contracts](#).

Versioning When Schema Validation Is Not Required

Strict schema compliance is rarely required. Many platforms tolerate extra elements not described by a schema. As long as this is tolerated, the full set of features described in [Data Contract Versioning](#) and [Forward-Compatible Data Contracts](#) can be used. The following guidelines are recommended.

Some of the guidelines must be followed exactly in order to send new versions of a type where an older one is expected or send an old one where the new one is expected. Other guidelines are not strictly required, but are listed here because they may be affected by the future of schema versioning.

1. Do not attempt to version data contracts by type inheritance. To create later versions, either change the data contract on an existing type or create a new unrelated type.
2. The use of inheritance together with data contracts is allowed, provided that inheritance is not used as a versioning mechanism and that certain rules are followed. If a type derives from a certain base type, do not make it derive from a different base type in a future version (unless it has the same data contract). There is one exception to this: you can insert a type into the hierarchy between a data contract type and its base type, but only if it does not contain data members with the same names as other members in any possible versions of the other types in the hierarchy. In general, using data members with the same names at different levels of the same inheritance hierarchy can lead to serious versioning problems and should be avoided.
3. Starting with the first version of a data contract, always implement [IExtensibleDataObject](#) to enable round-tripping. For more information, see [Forward-Compatible Data Contracts](#). If you have released one or more versions of a type without implementing this interface, implement it in the next version of the type.
4. In later versions, do not change the data contract name or namespace. If changing the name or namespace of the type underlying the data contract, be sure to preserve the data contract name and namespace by using the appropriate mechanisms, such as the [Name](#) property of the [DataContractAttribute](#). For more information about naming, see [Data Contract Names](#).
5. In later versions, do not change the names of any data members. If changing the name of the field, property, or event underlying the data member, use the [Name](#) property of the [DataMemberAttribute](#) to preserve the existing data member name.
6. In later versions, do not change the type of any field, property, or event underlying a data member such that the resulting data contract for that data member changes. Keep in mind that interface types are equivalent to [Object](#) for the purposes of determining the expected data contract.
7. In later versions, do not change the order of the existing data members by adjusting the [Order](#) property of the [DataMemberAttribute](#) attribute.
8. In later versions, new data members can be added. They should always follow these rules:
 - a. The [IsRequired](#) property should always be left at its default value of `false`.

- b. If a default value of `null` or zero for the member is unacceptable, a callback method should be provided using the [OnDeserializingAttribute](#) to provide a reasonable default in case the member is not present in the incoming stream. For more information about the callback, see [Version-Tolerant Serialization Callbacks](#).
 - c. The `Order` property on the `DataMemberAttribute` should be used to make sure that all of the newly added data members appear after the existing data members. The recommended way of doing this is as follows: None of the data members in the first version of the data contract should have their `Order` property set. All of the data members added in version 2 of the data contract should have their `Order` property set to 2. All of the data members added in version 3 of the data contract should have their `Order` set to 3, and so on. It is permissible to have more than one data member set to the same `Order` number.
- 9. Do not remove data members in later versions, even if the [IsRequired](#) property was left at its default property of `false` in prior versions.
 - 10. Do not change the `IsRequired` property on any existing data members from version to version.
 - 11. For required data members (where `IsRequired` is `true`), do not change the `EmitDefaultValue` property from version to version.
 - 12. Do not attempt to create branched versioning hierarchies. That is, there should always be a path in at least one direction from any version to any other version using only the changes permitted by these guidelines.

For example, if version 1 of a Person data contract contains only the Name data member, you should not create version 2a of the contract adding only the Age member and version 2b adding only the Address member. Going from 2a to 2b would involve removing Age and adding Address; going in the other direction would entail removing Address and adding Age. Removing members is not permitted by these guidelines.

- 13. You should generally not create new subtypes of existing data contract types in a new version of your application. Likewise, you should not create new data contracts that are used in place of data members declared as Object or as interface types. Creating these new classes is allowed only when you know that you can add the new types to the known types list of all instances of your old application. For example, in version 1 of your application, you may have the LibraryItem data contract type with the Book and Newspaper data contract subtypes. LibraryItem would then have a known types list that contains Book and Newspaper. Suppose you now add a Magazine type in version 2 which is a subtype of LibraryItem. If you send a Magazine instance from version 2 to version 1, the Magazine data contract is not found in the list of known types and an exception is thrown.
- 14. You should not add or remove enumeration members between versions. You should also not rename enumeration members, unless you use the Name property on the `EnumMemberAttribute` attribute to keep their names in the data contract model the same.
- 15. Collections are interchangeable in the data contract model as described in [Collection Types in Data Contracts](#). This allows for a great degree of flexibility. However, make sure that you do not inadvertently change a collection type in a non-interchangeable way from version to version. For example, do not change from a non-customized collection (that is, without the `CollectionDataContractAttribute` attribute) to a customized one or a customized collection to a non-customized one. Also, do not change the properties on the `CollectionDataContractAttribute` from version to version. The only allowed change is adding a Name or Namespace property if the underlying collection type's name or namespace has changed and you need to make its data contract name and namespace the same as in a previous version.

Some of the guidelines listed here can be safely ignored when special circumstances apply. Make sure you fully understand the serialization, deserialization, and schema mechanisms involved before deviating from the guidelines.

See Also

[Name](#)

[DataContractAttribute](#)

[Order](#)

[IsRequired](#)

[IExtensibleDataObject](#)

[ServiceBehaviorAttribute](#)

[ExtensionData](#)

[ExtensionDataObject](#)

[OnDeserializingAttribute](#)

[Using Data Contracts](#)

[Data Contract Versioning](#)

[Data Contract Names](#)

[Forward-Compatible Data Contracts](#)

[Version-Tolerant Serialization Callbacks](#)

Best Practices: Intermediaries

10/30/2018 • 2 minutes to read • [Edit Online](#)

Care must be taken to handle faults correctly when calling intermediaries to make sure service side channels on the intermediary are closed properly.

Consider the following scenario. A client makes a call to an intermediary which then calls a back-end service. The back-end service defines no fault contract, so any fault thrown from that service will be treated as an un-typed fault. The back-end service throws an [ApplicationException](#) and WCF correctly aborts the service-side channel. The [ApplicationException](#) then surfaces as a [FaultException](#) that is thrown to the intermediary. The intermediary re-throws the [ApplicationException](#). WCF interprets this as an un-typed fault from the intermediary and forwards it on to the client. Upon receiving the fault, both the intermediary and the client fault their client-side channels. The intermediary's service-side channel however remains open because WCF doesn't know the fault is fatal.

The best practice in this scenario is to detect if the fault coming from the service is fatal and if so the intermediary should fault its service-side channel as shown in the following code snippet.

```
catch (Exception e)
{
    bool faulted = service.State == CommunicationState.Faulted;
    service.Abort();
    if (faulted)
    {
        throw new ApplicationException(e.Message);
    }
    else
    {
        throw;
    }
}
```

See Also

[WCF Error Handling](#)

[Specifying and Handling Faults in Contracts and Services](#)

Service Versioning

5/5/2018 • 12 minutes to read • [Edit Online](#)

After initial deployment, and potentially several times during their lifetime, services (and the endpoints they expose) may need to be changed for a variety of reasons, such as changing business needs, information technology requirements, or to address other issues. Each change introduces a new version of the service. This topic explains how to consider versioning in Windows Communication Foundation (WCF).

Four Categories of Service Changes

The changes to services that may be required can be classified into four categories:

- **Contract changes:** For example, an operation might be added, or a data element in a message might be added or changed.
- **Address changes:** For example, a service moves to a different location where endpoints have new addresses.
- **Binding changes:** For example, a security mechanism changes or its settings change.
- **Implementation changes:** For example, when an internal method implementation changes.

Some of these changes are called "breaking" and others are "nonbreaking." A change is *nonbreaking* if all messages that would have been processed successfully in the previous version are processed successfully in the new version. Any change that does not meet that criterion is a *breaking* change.

Service Orientation and Versioning

One of the tenets of service orientation is that services and clients are autonomous (or independent). Among other things, this implies that service developers cannot assume that they control or even know about all service clients. This eliminates the option of rebuilding and redeploying all clients when a service changes versions. This topic assumes the service adheres to this tenet and therefore must be changed or "versioned" independent of its clients.

In cases where a breaking change is unexpected and cannot be avoided, an application may choose to ignore this tenet and require that clients be rebuilt and redeployed with a new version of the service.

Contract Versioning

Contracts used by a client do not need to be the same as the contract used by the service; they need only to be compatible.

For service contracts, compatibility means new operations exposed by the service can be added but existing operations cannot be removed or changed semantically.

For data contracts, compatibility means new schema type definitions can be added but existing schema type definitions cannot be changed in breaking ways. Breaking changes might include removing data members or changing their data type incompatibly. This feature allows the service some latitude in changing the version of its contracts without breaking clients. The next two sections explain nonbreaking and breaking changes that can be made to WCF data and service contracts.

Data Contract Versioning

This section deals with data versioning when using the [DataContractSerializer](#) and [DataContractAttribute](#) classes.

Strict Versioning

In many scenarios when changing versions is an issue, the service developer does not have control over the clients and therefore cannot make assumptions about how they would react to changes in the message XML or schema. In these cases, you must guarantee that the new messages will validate against the old schema, for two reasons:

- The old clients were developed with the assumption that the schema will not change. They may fail to process messages that they were never designed for.
- The old clients may perform actual schema validation against the old schema before even attempting to process the messages.

The recommended approach in such scenarios is to treat existing data contracts as immutable and create new ones with unique XML qualified names. The service developer would then either add new methods to an existing service contract or create a new service contract with methods that use the new data contract.

It will often be the case that a service developer needs to write some business logic that should run within all versions of a data contract plus version-specific business code for each version of the data contract. The appendix at the end of this topic explains how interfaces can be used to satisfy this need.

Lax Versioning

In many other scenarios, the service developer can make the assumption that adding a new, optional member to the data contract will not break existing clients. This requires the service developer to investigate whether existing clients are not performing schema validation and that they ignore unknown data members. In these scenarios, it is possible to take advantage of data contract features for adding new members in a nonbreaking way. The service developer can make this assumption with confidence if the data contract features for versioning were already used for the first version of the service.

WCF, ASP.NET Web Services, and many other Web service stacks support *lax versioning*: that is, they do not throw exceptions for new unknown data members in received data.

It is easy to mistakenly believe that adding a new member will not break existing clients. If you are unsure that all clients can handle lax versioning, the recommendation is to use the strict versioning guidelines and treat data contracts as immutable.

For detailed guidelines for both lax and strict versioning of data contracts, see [Best Practices: Data Contract Versioning](#).

Distinguishing Between Data Contract and .NET Types

A .NET class or structure can be projected as a data contract by applying the [DataContractAttribute](#) attribute to the class. The .NET type and its data contract projections are two distinct matters. It is possible to have multiple .NET types with the same data contract projection. This distinction is especially useful in allowing you to change the .NET type while maintaining the projected data contract, thereby maintaining compatibility with existing clients even in the strict sense of the word. There are two things you should always do to maintain this distinction between .NET type and data contract:

- Specify a [Name](#) and [Namespace](#). You should always specify the name and namespace of your data contract to prevent your .NET type's name and namespace from being exposed in the contract. This way, if you decide later to change the .NET namespace or type name, your data contract remains the same.
- Specify [Name](#). You should always specify the name of your data members to prevent your .NET member name from being exposed in the contract. This way, if you decide later to change the .NET name of the member, your data contract remains the same.

Changing or Removing Members

Changing the name or data type of a member, or removing data members is a breaking change even if lax versioning is allowed. If this is necessary, create a new data contract.

If service compatibility is of high importance, you might consider ignoring unused data members in your code and leave them in place. If you are splitting up a data member into multiple members, you might consider leaving the existing member in place as a property that can perform the required splitting and re-aggregation for down-level clients (clients that are not upgraded to the latest version).

Similarly, changes to the data contract's name or namespace are breaking changes.

Round-Trips of Unknown Data

In some scenarios, there is a need to "round-trip" unknown data that comes from members added in a new version. For example, a "versionNew" service sends data with some newly added members to a "versionOld" client. The client ignores the newly added members when processing the message, but it resends that same data, including the newly added members, back to the versionNew service. The typical scenario for this is data updates where data is retrieved from the service, changed, and returned.

To enable round-tripping for a particular type, the type must implement the [IExtensibleDataObject](#) interface. The interface contains one property, [ExtensionData](#) that returns the [ExtensionDataObject](#) type. The property is used to store any data from future versions of the data contract that is unknown to the current version. This data is opaque to the client, but when the instance is serialized, the content of the [ExtensionData](#) property is written with the rest of the data contract members' data.

It is recommended that all your types implement this interface to accommodate new and unknown future members.

Data Contract Libraries

There may be libraries of data contracts where a contract is published to a central repository, and service and type implementers implement and expose data contracts from that repository. In that case, when you publish a data contract to the repository, you have no control over who creates types that implement it. Thus, you cannot modify the contract once it is published, rendering it effectively immutable.

When Using the XmlSerializer

The same versioning principles apply when using the [XmlSerializer](#) class. When strict versioning is required, treat data contracts as immutable and create new data contracts with unique, qualified names for the new versions. When you are sure that lax versioning can be used, you can add new serializable members in new versions but not change or remove existing members.

NOTE

The [XmlSerializer](#) uses the [XmlAnyElementAttribute](#) and [XmlAnyAttributeAttribute](#) attributes to support round-tripping of unknown data.

Message Contract Versioning

The guidelines for message contract versioning are very similar to versioning data contracts. If strict versioning is required, you should not change your message body but instead create a new message contract with a unique qualified name. If you know that you can use lax versioning, you can add new message body parts but not change or remove existing ones. This guidance applies both to bare and wrapped message contracts.

Message headers can always be added, even if strict versioning is in use. The `MustUnderstand` flag may affect versioning. In general, the versioning model for headers in WCF is as described in the SOAP specification.

Service Contract Versioning

Similar to data contract versioning, service contract versioning also involves adding, changing, and removing operations.

Specifying Name, Namespace, and Action

By default, the name of a service contract is the name of the interface. Its default namespace is "http://tempuri.org", and each operation's action is "http://tempuri.org/contractname/methodname". It is recommended that you explicitly specify a name and namespace for the service contract, and an action for each operation to avoid using "http://tempuri.org" and to prevent interface and method names from being exposed in the service's contract.

Adding Parameters and Operations

Adding service operations exposed by the service is a nonbreaking change because existing clients need not be concerned about those new operations.

NOTE

Adding operations to a duplex callback contract is a breaking change.

Changing Operation Parameter or Return Types

Changing parameter or return types generally is a breaking change unless the new type implements the same data contract implemented by the old type. To make such a change, add a new operation to the service contract or define a new service contract.

Removing Operations

Removing operations is also a breaking change. To make such a change, define a new service contract and expose it on a new endpoint.

Fault Contracts

The [FaultContractAttribute](#) attribute enables a service contract developer to specify information about faults that can be returned from the contract's operations.

The list of faults described in a service's contract is not considered exhaustive. At any time, an operation may return faults that are not described in its contract. Therefore changing the set of faults described in the contract is not considered breaking. For example, adding a new fault to the contract using the [FaultContractAttribute](#) or removing an existing fault from the contract.

Service Contract Libraries

Organizations may have libraries of contracts where a contract is published to a central repository and service implementers implement contracts from that repository. In this case, when you publish a service contract to the repository you have no control over who creates services that implement it. Therefore, you cannot modify the service contract once published, rendering it effectively immutable. WCF supports contract inheritance, which can be used to create a new contract that extends existing contracts. To use this feature, define a new service contract interface that inherits from the old service contract interface, then add methods to the new interface. You then change the service that implements the old contract to implement the new contract and change the "versionOld" endpoint definition to use the new contract. To "versionOld" clients, the endpoint will continue to appear as exposing the "versionOld" contract; to "versionNew" clients, the endpoint will appear to expose the "versionNew" contract.

Address and Binding Versioning

Changes to endpoint address and binding are breaking changes unless clients are capable of dynamically discovering the new endpoint address or binding. One mechanism for implementing this capability is by using a Universal Discovery Description and Integration (UDDI) registry and the UDDI Invocation Pattern where a client attempts to communicate with an endpoint and, upon failure, queries a well-known UDDI registry for the current endpoint metadata. The client then uses the address and binding from this metadata to communicate with the endpoint. If this communication succeeds, the client caches the address and binding information for future use.

Routing Service and Versioning

If the changes made to a service are breaking changes and you need to have two or more different versions of a service running simultaneously you can use the WCF Routing Service to route messages to the appropriate service instance. The WCF Routing Service uses content-based routing, in other words, it uses information within the message to determine where to route the message. For more information about the WCF Routing Service see [Routing Service](#). For an example of how to use the WCF Routing Service for service versioning see [How To: Service Versioning](#).

Appendix

The general data contract versioning guidance when strict versioning is needed is to treat data contracts as immutable and create new ones when changes are required. A new class needs to be created for each new data contract, so a mechanism is needed to avoid having to take existing code that was written in terms of the old data contract class and rewrite it in terms of the new data contract class.

One such mechanism is to use interfaces to define the members of each data contract and write internal implementation code in terms of the interfaces rather than the data contract classes that implement the interfaces. The following code for version 1 of a service shows an `IPurchaseOrderV1` interface and a `PurchaseOrderV1`:

```
public interface IPurchaseOrderV1
{
    string OrderId { get; set; }
    string CustomerId { get; set; }
}

[DataContract(
    Name = "PurchaseOrder",
    Namespace = "http://examples.microsoft.com/WCF/2005/10/PurchaseOrder")]
public class PurchaseOrderV1 : IPurchaseOrderV1
{
    [DataMember(...)]
    public string OrderId {...}
    [DataMember(...)]
    public string CustomerId {...}
}
```

While the service contract's operations would be written in terms of `PurchaseOrderV1`, the actual business logic would be in terms of `IPurchaseOrderV1`. Then, in version 2, there would be a new `IPurchaseOrderV2` interface and a new `PurchaseOrderV2` class as shown in the following code:

```
public interface IPurchaseOrderV2
{
    DateTime OrderDate { get; set; }
}

[DataContract(
    Name = "PurchaseOrder",
    Namespace = "http://examples.microsoft.com/WCF/2006/02/PurchaseOrder")]
public class PurchaseOrderV2 : IPurchaseOrderV1, IPurchaseOrderV2
{
    [DataMember(...)]
    public string OrderId {...}
    [DataMember(...)]
    public string CustomerId {...}
    [DataMember(...)]
    public DateTime OrderDate { ... }
}
```

The service contract would be updated to include new operations that are written in terms of `PurchaseOrderV2` . Existing business logic written in terms of `IPurchaseOrderV1` would continue to work for `PurchaseOrderV2` and new business logic that needs the `OrderDate` property would be written in terms of `IPurchaseOrderV2` .

See Also

[DataContractSerializer](#)

[DataContractAttribute](#)

[Name](#)

[Namespace](#)

[Order](#)

[IsRequired](#)

[IExtensibleDataObject](#)

[ExtensionDataObject](#)

[ExtensionData](#)

[XmlSerializer](#)

[Data Contract Equivalence](#)

[Version-Tolerant Serialization Callbacks](#)

Load Balancing

5/5/2018 • 3 minutes to read • [Edit Online](#)

One way to increase the capacity of Windows Communication Foundation (WCF) applications is to scale them out by deploying them into a load-balanced server farm. WCF applications can be load balanced using standard load balancing techniques, including software load balancers such as Windows Network Load Balancing as well as hardware-based load balancing appliances.

The following sections discuss considerations for load balancing WCF applications built using various system-provided bindings.

Load Balancing with the Basic HTTP Binding

From the perspective of load balancing, WCF applications that communicate using the [BasicHttpBinding](#) are no different than other common types of HTTP network traffic (static HTML content, ASP.NET pages, or ASMX Web Services). WCF channels that use this binding are inherently stateless, and terminate their connections when the channel closes. As such, the [BasicHttpBinding](#) works well with existing HTTP load balancing techniques.

By default, the [BasicHttpBinding](#) sends a connection HTTP header in messages with a `Keep-Alive` value, which enables clients to establish persistent connections to the services that support them. This configuration offers enhanced throughput because previously established connections can be reused to send subsequent messages to the same server. However, connection reuse may cause clients to become strongly associated to a specific server within the load-balanced farm, which reduces the effectiveness of round-robin load balancing. If this behavior is undesirable, HTTP `Keep-Alive` can be disabled on the server using the [KeepAliveEnabled](#) property with a [CustomBinding](#) or user-defined [Binding](#). The following example shows how to do this using configuration.

```

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

  <system.serviceModel>
    <services>
      <service
        name="Microsoft.ServiceModel.Samples.CalculatorService"
        behaviorConfiguration="CalculatorServiceBehavior">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8000/servicemodelsamples/service"/>
          </baseAddresses>
        </host>
        <!-- configure http endpoint, use base address provided by host
            And the customBinding -->
        <endpoint address=""
          binding="customBinding"
          bindingConfiguration="HttpBinding"
          contract="Microsoft.ServiceModel.Samples.ICalculator" />
      </service>
    </services>

    <bindings>
      <customBinding>

        <!-- Configure a CustomBinding that disables keepAliveEnabled-->
        <binding name="HttpBinding" keepAliveEnabled="False"/>

      </customBinding>
    </bindings>
  </system.serviceModel>
</configuration>

```

Using the simplified configuration introduced in .NET Framework 4, the same behavior can be accomplished using the following simplified configuration.

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <protocolMapping>
      <add scheme="http" binding="customBinding" />
    </protocolMapping>
    <bindings>
      <customBinding>

        <!-- Configure a CustomBinding that disables keepAliveEnabled-->
        <binding keepAliveEnabled="False"/>

      </customBinding>
    </bindings>
  </system.serviceModel>
</configuration>

```

For more information about default endpoints, bindings, and behaviors, see [Simplified Configuration](#) and [Simplified Configuration for WCF Services](#).

Load Balancing with the WSHttp Binding and the WSDualHttp Binding

Both the [WSHttpBinding](#) and the [WSDualHttpBinding](#) can be load balanced using HTTP load balancing techniques provided several modifications are made to the default binding configuration.

- Turn off Security Context Establishment: this can be accomplished by the setting the

[EstablishSecurityContext](#) property on the [WSHttpBinding](#) to `false`. Alternatively, if security sessions are required, it is possible to use stateful security sessions as described in the [Secure Sessions](#) topic. Stateful security sessions enable the service to remain stateless as all of the state for the security session is transmitted with each request as a part of the protection security token. Note that to enable a stateful security session, it is necessary to use a [CustomBinding](#) or user-defined [Binding](#) as the necessary configuration settings are not exposed on [WSHttpBinding](#) and [WSDualHttpBinding](#) that are provided by the system.

- Do not use reliable sessions. This feature is off by default.

Load Balancing the Net.TCP Binding

The [NetTcpBinding](#) can be load balanced using IP-layer load balancing techniques. However, the [NetTcpBinding](#) pools TCP connections by default to reduce connection latency. This is an optimization that interferes with the basic mechanism of load balancing. The primary configuration value for optimizing the [NetTcpBinding](#) is the lease timeout, which is part of the Connection Pool Settings. Connection pooling causes client connections to become associated to specific servers within the farm. As the lifetime of those connections increase (a factor controlled by the lease timeout setting), the load distribution across various servers in the farm becomes unbalanced. As a result the average call time increases. So when using the [NetTcpBinding](#) in load-balanced scenarios, consider reducing the default lease timeout used by the binding. A 30-second lease timeout is a reasonable starting point for load-balanced scenarios, although the optimal value is application-dependent. For more information about the channel lease timeout and other transport quotas, see [Transport Quotas](#).

For best performance in load-balanced scenarios, consider using [NetTcpSecurity](#) (either [Transport](#) or [TransportWithMessageCredential](#)).

See Also

[Internet Information Services Hosting Best Practices](#)

Controlling Resource Consumption and Improving Performance

8/31/2018 • 4 minutes to read • [Edit Online](#)

This topic describes various properties in different areas of the Windows Communication Foundation (WCF) architecture that work to control resource consumption and affect performance metrics.

Properties that Constrain Resource Consumption in WCF

Windows Communication Foundation (WCF) applies constraints on certain types of processes for either security or performance purposes. These constraints come in two main forms, either quotas and throttles. *Quotas* are limits that when reached or exceeded trigger an immediate exception at some point in the system. *Throttles* are limits that do not immediately cause an exception to be thrown. Instead, when a throttle limit is reached, processing continues but within the limits set by that throttle value. This limited processing might trigger an exception elsewhere, but this depends upon the application.

In addition to the distinction between quotas and throttles, some constraining properties are located at the serialization level, some at the transport level, and some at the application level. For example, the quota [TransportBindingElement.MaxReceivedMessageSize](#), which is implemented by all system-supplied transport binding elements, is set to 65,536 bytes by default to hinder malicious clients from engaging in denial-of-service attacks against a service by causing excessive memory consumption. (Typically, you can increase performance by lowering this value.)

An example of a serialization quota is the [DataContractSerializer.MaxItemsInObjectGraph](#) property, which specifies the maximum number of objects that the serializer serializes or deserializes in a single [ReadObject](#) method call. An example of an application-level throttle is the [ServiceThrottle.MaxConcurrentSessions](#) property, which by default restricts the number of concurrent sessionful channel connections to 10. (Unlike the quotas, if this throttle value is reached, the application continues processing but accepts no new sessionful channels, which means that new clients cannot connect until one of the other sessionful channels is ended.)

These controls are designed to provide an out-of-the-box mitigation against certain types of attacks or to improve performance metrics such as memory footprint, start-up time, and so on. However, depending on the application, these controls can impede service application performance or prevent the application from working at all. For example, an application designed to stream video can easily exceed the default [TransportBindingElement.MaxReceivedMessageSize](#) property. This topic provides an overview of the various controls applied to applications at all levels of WCF, describes various ways to obtain more information about whether a setting is hindering your application, and describes ways to correct various problems. Most throttles and some quotas are available at the application level, even when the base property is a serialization or transport constraint. For example, you can set the [DataContractSerializer.MaxItemsInObjectGraph](#) property using the [ServiceBehaviorAttribute.MaxItemsInObjectGraph](#) property on the service class.

NOTE

If you have a particular problem, you should first read the [WCF Troubleshooting Quickstart](#) to see whether your problem (and a solution) is listed there.

Properties that restrict serialization processes are listed in [Security Considerations for Data](#). Properties that restrict the consumption of resources related to transports are listed in [Transport Quotas](#). Properties that restrict the consumption of resources at the application layer are the members of the [ServiceThrottle](#) class.

Detecting Application and Performance Issues Related to Quota Settings

The defaults of the preceding values have been chosen to enable basic application functionality across a wide range of application types while providing basic protection against common security issues. However, different application designs might exceed one or more throttle settings although the application otherwise is secure and would work as designed. In these cases, you must identify which throttle values are being exceeded and at what level, and decide on the appropriate course of action to increase application throughput.

Typically, when writing the application and debugging it, you set the [ServiceDebugBehavior.IncludeExceptionDetailInFaults](#) property to `true` in the configuration file or programmatically. This instructs WCF to return service exception stack traces to the client application for viewing. This feature reports most application-level exceptions in such a way as to display which quota settings might be involved, if that is the problem.

Some exceptions happen at run time below the visibility of the application layer and are not returned using this mechanism, and they might not be handled by a custom [System.ServiceModel.Dispatcher.IErrorHandler](#) implementation. If you are in a development environment like Microsoft Visual Studio, most of these exceptions are displayed automatically. However, some exceptions can be masked by development environment settings such as [Just My Code](#) Visual Studio.

Regardless of the capabilities of your development environment, you can use capabilities of WCF tracing and message logging to debug all exceptions and tune the performance of your applications. For more information, see [Using Tracing to Troubleshoot Your Application](#).

Performance Issues and XmlSerializer

Services and client applications that use data types that are serializable using the [XmlSerializer](#) generate and compile serialization code for those data types at run time, which can result in slow start-up performance.

NOTE

Pre-generated serialization code can be used only in client applications and not in services.

The [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) can improve start-up performance for these applications by generating the necessary serialization code from the compiled assemblies for the application. For more information, see [How to: Improve the Startup Time of WCF Client Applications using the XmlSerializer](#).

Performance Issues When Hosting WCF Services Under ASP.NET

When a WCF service is hosted under IIS and ASP.NET, the configuration settings of IIS and ASP.NET can affect the throughput and memory footprint of the WCF service. For more information about ASP.NET performance, see [Improving ASP.NET Performance](#). One setting that might have unintended consequences is [MinWorkerThreads](#), which is a property of the [ProcessModelSection](#). If your application has a fixed or small number of clients, setting [MinWorkerThreads](#) to 2 might provide a throughput boost on a multiprocessor machine that has a CPU utilization close to 100%. This increase in performance comes with a cost: it will also cause an increase in memory usage, which could reduce scalability.

See Also

- [Administration and Diagnostics](#)
- [Large Data and Streaming](#)

Deploying WCF Applications with ClickOnce

8/31/2018 • 2 minutes to read • [Edit Online](#)

Client applications using Windows Communication Foundation (WCF) may be deployed using ClickOnce technology. This technology allows them to take advantage of the runtime security protections provided by Code Access Security, provided that they are digitally signed with a trusted certificate. The certificate used to sign the ClickOnce application must reside in the Trusted Publisher store, and the local security policy on the client machine must be configured to grant Full Trust permissions to applications signed with the publisher's certificate.

For information on configuring ClickOnce applications and trusted publishers, see [Configuring ClickOnce Trusted Publishers](#).

See Also

[Trusted Application Deployment Overview](#)

[ClickOnce Deployment for Windows Forms Applications](#)

Administration and Diagnostics

5/5/2018 • 2 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) provides a rich set of functionalities that can help you monitor the different stages of an application's life. For example, you can use configuration to set up services and clients at deployment. WCF includes a large set of performance counters to help you gauge your application's performance. WCF also exposes inspection data of a service at runtime through a WCF Windows Management Instrumentation (WMI) provider. When the application experiences a failure or starts acting improperly, you can use the Event Log to see if anything significant has occurred. You can also use Message Logging and Tracing to see what events are happening end-to-end in your application. These features assist both developers and IT professionals to troubleshoot an WCF application when it is not behaving correctly.

NOTE

If you are receiving faults with no specific detail information, you should enable the `includeExceptionDetailInFaults` attribute of the `<serviceDebug>` configuration element. This instructs WCF to send exception detail to clients, which enables you to detect many common problems without requiring more advanced diagnosis. For more information, see [Sending and Receiving Faults](#).

Diagnostics Features Provided by WCF

WCF provides the following diagnostics functionalities:

- End-To-End tracing provides instrumentation data for troubleshooting an application without using a debugger. WCF outputs traces for process milestones, as well as error messages. This can include opening a channel factory or sending and receiving messages by a service host. Tracing can be enabled for a running application to monitor its progress. For more information, see the [Tracing](#) topic. To understand how you can use tracing to debug your application, see the [Using Tracing to Troubleshoot Your Application](#) topic.
- Message logging allows you to see how messages look both before and after transmission. For more information, see the [Message Logging](#) topic.
- Event tracing writes events in the Event Log for any major issues. You can then use the Event Viewer to examine any abnormalities. For more information, see the [Event Logging](#) topic.
- Performance counters exposed through Performance Monitor enable you to monitor your application and system's health. For more information, see the [Performance Counters](#) topic.
- The [System.ServiceModel.Configuration](#) namespace allows you to load configuration files and set up a service or client endpoint. You can use the object model to script changes to many applications when updates must be deployed to many computers. Alternatively, you can use the [Configuration Editor Tool \(SvcConfigEditor.exe\)](#) to edit the configuration settings using a GUI wizard. For more information, see the [Configuring Your Application](#) topic.
- WMI enables you to find out what services are listening on a machine and the bindings that are in use. For more information, see the [Using Windows Management Instrumentation for Diagnostics](#) topic.

WCF also provides several GUI and command line tools to make it easier for you to create, deploy, and manage WCF applications. For more information, see [Windows Communication Foundation Tools](#). For example, you can use the [Configuration Editor Tool \(SvcConfigEditor.exe\)](#) to create and edit WCF configuration settings using a wizard, instead of editing XML directly. You can also use the [Service Trace Viewer Tool \(SvcTraceViewer.exe\)](#) to

view, group, and filter trace messages so that you can diagnose, repair, and verify issues with WCF services.

See Also

[Configuring Your Application](#)

[Deploying Services](#)

[Exceptions Reference](#)

[Event Logging](#)

[Message Logging](#)

[Configuration Editor Tool \(SvcConfigEditor.exe\)](#)

[Service Trace Viewer Tool \(SvcTraceViewer.exe\)](#)

[ServiceModel Registration Tool](#)

[Tracing](#)

[Using Windows Management Instrumentation for Diagnostics](#)

[Performance Counters](#)

[Windows Communication Foundation Tools](#)

WCF System Requirements

5/5/2018 • 2 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) is a communication infrastructure that is used to create distributed applications. The following lists the requirements that enable WCF to run.

System Requirements

WCF is installed by default on Windows Vista.

WCF can also be installed on Windows XP SP2, Windows Server 2003 R2, or Windows Server 2003 SP1.

Note The Message Queuing (MSMQ) functionality of WCF is supported only on Windows Vista, Windows Server 2003 R2, Windows Server 2003 SP1, and Windows XP Professional.

See Also

[Conceptual Overview](#)

[Basic WCF Programming](#)

[WCF Feature Details](#)

[Guidelines and Best Practices](#)

Operating System Resources Required by WCF

5/4/2018 • 2 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) depends on several resources that are provided by the operating system to function. The following table lists those resources.

RESOURCE	DESCRIPTION
Microsoft Distributed Transaction Coordinator (MSDTC)	Required to support OleTx transactions.
Internet Information Services (IIS)	Required if you want to use IIS to host your application.
Windows Process Activation Service (WAS)	Required if you want to use WAS to host your application.

See Also

[System Requirements](#)

Troubleshooting Setup Issues

10/3/2018 • 2 minutes to read • [Edit Online](#)

This topic describes how to troubleshoot Windows Communication Foundation (WCF) set up issues.

Some Windows Communication Foundation Registry Keys are not Repaired by Performing an MSI Repair Operation on the .NET Framework 3.0

If you delete any of the following registry keys:

- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\ServiceModelService 3.0.0.0
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\ServiceModelOperation 3.0.0.0
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\ServiceModelEndpoint 3.0.0.0
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SMSvcHost 3.0.0.0
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\MSDTC Bridge 3.0.0.0

The keys are not re-created if you run repair by using the .NET Framework 3.0 installer launched from the **Add/Remove Programs** applet in **Control Panel**. To recreate these keys correctly, the user must uninstall and reinstall the .NET Framework 3.0.

WMI Service Corruption Blocks Installation of the Windows Communication Foundation WMI provider during installation of .NET Framework 3.0 package

WMI Service Corruption may block the installation of the Windows Communication Foundation WMI provider. During installation the Windows Communication Foundation installer is unable to register the WCF .mof file using the mofcomp.exe component. The following is a list of symptoms:

1. .NET Framework 3.0 installation completes successfully, but the WCF WMI provider is not registered.
2. An error event appears in the application event log that references problems registering the WMI provider for WCF, or running mofcomp.exe.
3. The setup log file named dd_wcf_retCA* in the user's %temp% directory contains references to failure to register the WCF WMI provider.
4. An exception such as one the following may be listed in the event log or setup trace log file:

```
ServiceModelReg [11:09:59:046]: System.ApplicationException: Unexpected result 3 executing  
E:\WINDOWS\system32\wbem\mofcomp.exe with  
"E:\WINDOWS\Microsoft.NET\Framework\v3.0\Windows Communication Foundation\ServiceModel.mof"
```

or:

```
ServiceModelReg [07:19:33:843]: System.TypeInitializationException: The type initializer for  
'System.Management.ManagementPath' threw an exception. --->  
System.Runtime.InteropServices.COMException (0x80040154): Retrieving the COM class factory for  
component with CLSID {CF4CC405-E2C5-4DDD-B3CE-5E7582D8C9FA} failed due to the following error:
```

80040154.

or:

ServiceModelReg [07:19:32:750]: System.IO.FileNotFoundException: Could not load file or assembly 'C:\WINDOWS\system32\wbem\mofcomp.exe' or one of its dependencies. The system cannot find the file specified.

File name: 'C:\WINDOWS\system32\wbem\mofcomp.exe'

The following steps must be followed to resolve the problem described previously.

1. Run [the WMI Diagnosis Utility, version 2.0](#) to repair the WMI service. For more information about using this tool, see the [WMI Diagnosis Utility](#) topic.

Repair the .NET Framework 3.0 installation by using the **Add/Remove Programs** applet located in **Control Panel**, or uninstall/reinstall the .NET Framework 3.0.

Repairing .NET Framework 3.0 after .NET Framework 3.5 Installation Removes Configuration Elements Introduced by .NET Framework 3.5 in machine.config

If you do a repair of .NET Framework 3.0 after you installed .NET Framework 3.5, configuration elements introduced by .NET Framework 3.5 in machine.config are removed. However, the web.config remains intact. The workaround is to repair .NET Framework 3.5 after this via ARP, or use the [WorkFlow Service Registration Tool \(WFServicesReg.exe\)](#) with the `/c` switch.

[WorkFlow Service Registration Tool \(WFServicesReg.exe\)](#) can be found at %windir%\Microsoft.NET\Framework\v3.5\ or %windir%\Microsoft.NET\Framework64\v3.5\

Configure IIS Properly for WCF/WF Webhost after Installing .NET Framework 3.5

When .NET Framework 3.5 installation fails to configure additional WCF-related IIS configuration settings, it logs an error in the installation log and continues. Any attempt to run WorkflowServices applications will fail, since the required configuration settings are missing. For example, loading xoml or rules service can fail.

To workaround this problem, use the [WorkFlow Service Registration Tool \(WFServicesReg.exe\)](#) with the `/c` switch to properly configure IIS script maps on the machine. [WorkFlow Service Registration Tool \(WFServicesReg.exe\)](#) can be found at %windir%\Microsoft.NET\Framework\v3.5\ or %windir%\Microsoft.NET\Framework64\v3.5\

Could not load type 'System.ServiceModel.Activation.HttpModule' from assembly 'System.ServiceModel, Version 3.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089'

This error occurs if .NET Framework 4 is installed and then WCF HTTP Activation is enabled. To resolve the issue run the following command-line from inside the Developer Command Prompt for Visual Studio:

```
aspnet_regiis.exe -i -enable
```

See Also

[Set-Up Instructions](#)

Migrating from .NET Remoting to WCF

10/19/2018 • 20 minutes to read • [Edit Online](#)

This article describes how to migrate an application that uses .NET Remoting to use Windows Communication Foundation (WCF). It compares similar concepts between these products and then describes how to accomplish several common Remoting scenarios in WCF.

.NET Remoting is a legacy product that is supported only for backward compatibility. It is not secure across mixed-trust environments because it cannot maintain the separate trust levels between client and server. For example, you should never expose a .NET Remoting endpoint to the Internet or to untrusted clients. We recommend existing Remoting applications be migrated to newer and more secure technologies. If the application's design uses only HTTP and is RESTful, we recommend ASP.NET Web API. For more information, see ASP.NET Web API. If the application is based on SOAP or requires non-Http protocols such as TCP, we recommend WCF.

Comparing .NET Remoting to WCF

This section compares the basic building blocks of .NET Remoting with their WCF equivalents. We will use these building blocks later to create some common client-server scenarios in WCF. The following chart summarizes the main similarities and differences between .NET Remoting and WCF.

	.NET REMOTING	WCF
Server type	Subclass MarshalByRefObject	Mark with [ServiceContract] attribute
Service operations	Public methods on server type	Mark with [OperationContract] attribute
Serialization	ISerializable or [Serializable]	DataContractSerializer or XmlSerializer
Objects passed	By-value or by-reference	By-value only
Errors/exceptions	Any serializable exception	FaultContract<TDetail>
Client proxy objects	Strongly typed transparent proxies are created automatically from MarshalByRefObjects	Strongly typed proxies are generated on-demand using ChannelFactory<TChannel>
Platform required	Both client and server must use Microsoft OS and .NET	Cross-platform
Message format	Private	Industry standards (SOAP, WS-*, etc.)

Server Implementation Comparison

Creating a Server in .NET Remoting

.NET Remoting server types must derive from MarshalByRefObject and define methods the client can call, like the following:

```
public class RemotingServer : MarshalByRefObject
{
    public Customer GetCustomer(int customerId) { ... }
}
```


The public methods of this server type become the public contract available to clients. There is no separation between the server's public interface and its implementation – one type handles both.

Once the server type has been defined, it can be made available to clients, like in the following example:

```
TcpChannel channel = new TcpChannel(8080);
ChannelServices.RegisterChannel(channel, ensureSecurity : true);
RemotingConfiguration.RegisterWellKnownServiceType(
    typeof(RemotingServer),
    "RemotingServer",
    WellKnownObjectMode.Singleton);
Console.WriteLine("RemotingServer is running. Press ENTER to terminate...");
Console.ReadLine();
```

There are many ways to make the Remoting type available as a server, including using configuration files. This is just one example.

Creating a Server in WCF

The equivalent step in WCF involves creating two types -- the public "service contract" and the concrete implementation. The first is declared as an interface marked with [ServiceContract]. Methods available to clients are marked with [OperationContract]:

```
[ServiceContract]
public interface IWCFServer
{
    [OperationContract]
    Customer GetCustomer(int customerId);
}
```

The server's implementation is defined in a separate concrete class, like in the following example:

```
public class WCFServer : IWCFServer
{
    public Customer GetCustomer(int customerId) { ... }
}
```

Once these types have been defined, the WCF server can be made available to clients, like in the following example:

```
NetTcpBinding binding = new NetTcpBinding();
Uri baseAddress = new Uri("net.tcp://localhost:8080/wcfserver");

using (ServiceHost serviceHost = new ServiceHost(typeof(WCFServer), baseAddress))
{
    serviceHost.AddServiceEndpoint(typeof(IWCFServer), binding, baseAddress);
    serviceHost.Open();

    Console.WriteLine(String.Format("The WCF server is ready at {0}.",
                                    baseAddress));
    Console.WriteLine("Press <ENTER> to terminate service...");
    Console.WriteLine();
    Console.ReadLine();
}
```

NOTE

TCP is used in both examples to keep them as similar as possible. Refer to the scenario walk-throughs later in this topic for examples using HTTP.

There are many ways to configure and to host WCF services. This is just one example, known as "self-hosted". For more information, see the following topics:

- [How to: Define a Service Contract](#)
- [Configuring Services Using Configuration Files](#)
- [Hosting Services](#)

Client Implementation Comparison

Creating a Client in .NET Remoting

Once a .NET Remoting server object has been made available, it can be consumed by clients, like in the following example:

```
TcpChannel channel = new TcpChannel();
ChannelServices.RegisterChannel(channel, ensureSecurity : true);
RemotingServer server = (RemotingServer)Activator.GetObject(
    typeof(RemotingServer),
    "tcp://localhost:8080/RemotingServer");

RemotingCustomer customer = server.GetCustomer(42);
Console.WriteLine(String.Format("Customer {0} {1} received.",
    customer.FirstName, customer.LastName));
```

The RemotingServer instance returned from Activator.GetObject() is known as a "transparent proxy." It implements the public API for the RemotingServer type on the client, but all the methods call the server object running in a different process or machine.

Creating a Client in WCF

The equivalent step in WCF involves using a channel factory to create the proxy explicitly. Like Remoting, the proxy object can be used to invoke operations on the server, like in the following example:

```
NetTcpBinding binding = new NetTcpBinding();
String url = "net.tcp://localhost:8000/wcfserver";
EndpointAddress address = new EndpointAddress(url);
ChannelFactory<IWCFServer> channelFactory =
    new ChannelFactory<IWCFServer>(binding, address);
IWCFServer server = channelFactory.CreateChannel();

Customer customer = server.GetCustomer(42);
Console.WriteLine(String.Format("Customer {0} {1} received.",
    customer.FirstName, customer.LastName));
```

This example shows programming at the channel level because it is most similar to the Remoting example. Also available is the **Add Service Reference** approach in Visual Studio that generates code to simplify client programming. For more information, see the following topics:

- [Client Channel-Level Programming](#)
- [How to: Add, Update, or Remove a Service Reference](#)

Serialization Usage

Both .NET Remoting and WCF use serialization to send objects between client and server, but they differ in these

important ways:

1. They use different serializers and conventions to indicate what to serialize.
2. .NET Remoting supports "by reference" serialization that allows method or property access on one tier to execute code on the other tier, which is across security boundaries. This capability exposes security vulnerabilities and is one of the main reasons why Remoting endpoints should never be exposed to untrusted clients.
3. Serialization used by Remoting is opt-out (explicitly exclude what not to serialize) and WCF serialization is opt-in (explicitly mark which members to serialize).

Serialization in .NET Remoting

.NET Remoting supports two ways to serialize and deserialize objects between the client and server:

- *By value* – the values of the object are serialized across tier boundaries, and a new instance of that object is created on the other tier. Any calls to methods or properties of that new instance execute only locally and do not affect the original object or tier.
- *By reference* – a special "object reference" is serialized across tier boundaries. When one tier interacts with methods or properties of that object, it communicates back to the original object on the original tier. By-reference objects can flow in either direction – server to client, or client to server.

By-value types in Remoting are marked with the [Serializable] attribute or implement ISerializable, like in the following example:

```
[Serializable]
public class RemotingCustomer
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int CustomerId { get; set; }
}
```

By-reference types derive from the MarshalByRefObject class, like in the following example:

```
public class RemotingCustomerReference : MarshalByRefObject
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int CustomerId { get; set; }
}
```

It is extremely important to understand the implications of Remoting's by-reference objects. If either tier (client or server) sends a by-reference object to the other tier, all method calls execute back on the tier owning the object. For example, a client calling methods on a by-reference object returned by the server will execute code on the server. Similarly, a server calling methods on a by-reference object provided by the client will execute code back on the client. For this reason, the use of .NET Remoting is recommended only within fully-trusted environments. Exposing a public .NET Remoting endpoint to untrusted clients will make a Remoting server vulnerable to attack.

Serialization in WCF

WCF supports only by-value serialization. The most common way to define a type to exchange between client and server is like in the following example:

```
[DataContract]
public class WCFCustomer
{
    [DataMember]
    public string FirstName { get; set; }

    [DataMember]
    public string LastName { get; set; }

    [DataMember]
    public int CustomerId { get; set; }
}
```

The [DataContract] attribute identifies this type as one that can be serialized and deserialized between client and server. The [DataMember] attribute identifies the individual properties or fields to serialize.

When WCF sends an object across tiers, it serializes only the values and creates a new instance of the object on the other tier. Any interactions with the values of the object occur only locally – they do not communicate with the other tier the way .NET Remoting by-reference objects do. For more information, see the following topics:

- [Serialization and Deserialization](#)
- [Serialization in Windows Communication Foundation](#)

Exception Handling Capabilities

Exceptions in .NET Remoting

Exceptions thrown by a Remoting server are serialized, sent to the client, and thrown locally on the client like any other exception. Custom exceptions can be created by sub-classing the Exception type and marking it with [Serializable]. Most framework exceptions are already marked in this way, allowing most to be thrown by the server, serialized, and re-thrown on the client. Though this design is convenient during development, server-side information can inadvertently be disclosed to the client. This is one of many reasons Remoting should be used only in fully-trusted environments.

Exceptions and Faults in WCF

WCF does not allow arbitrary exception types to be returned from the server to the client because it could lead to inadvertent information disclosure. If a service operation throws an unexpected exception, it causes a general purpose FaultException to be thrown on the client. This exception does not carry any information why or where the problem occurred, and for some applications this is sufficient. Applications that need to communicate richer error information to the client do this by defining a fault contract.

To do this, first create a [DataContract] type to carry the fault information.

```
[DataContract]
public class CustomerServiceFault
{
    [DataMember]
    public string ErrorMessage { get; set; }

    [DataMember]
    public int CustomerId {get;set;}
}
```

Specify the fault contract to use for each service operation.

```
[ServiceContract]
public interface IWCFServer
{
    [OperationContract]
    [FaultContract(typeof(CustomerServiceFault))]
    Customer GetCustomer(int customerId);
}
```

The server reports error conditions by throwing a `FaultException`.

```
throw new FaultException<CustomerServiceFault>(
    new CustomerServiceFault() {
        CustomerId = customerId,
        ErrorMessage = "Illegal customer Id"
    });
```

And whenever the client makes a request to the server, it can catch faults as normal exceptions.

```
try
{
    Customer customer = server.GetCustomer(-1);
}
catch (FaultException<CustomerServiceFault> fault)
{
    Console.WriteLine(String.Format("Fault received: {0}",
        fault.Detail.ErrorMessage));
}
```

For more information about fault contracts, see [FaultException](#).

Security Considerations

Security in .NET Remoting

Some .NET Remoting channels support security features such as authentication and encryption at the channel layer (IPC and TCP). The HTTP channel relies on Internet Information Services (IIS) for both authentication and encryption. Despite this support, you should consider .NET Remoting an unsecure communication protocol and use it only within fully-trusted environments. Never expose a public Remoting endpoint to the Internet or untrusted clients.

Security in WCF

WCF was designed with security in mind, in part to address the kinds of vulnerabilities found in .NET Remoting. WCF offers security at both the transport and message level, and offers many options for authentication, authorization, encryption, and so on. For more information, see the following topics:

- [Security](#)
- [WCF Security Guidance](#)

Migrating to WCF

Why Migrate from Remoting to WCF?

- **.NET Remoting is a legacy product.** As described in [.NET Remoting](#), it is considered a legacy product and is not recommended for new development. WCF or ASP.NET Web API are recommended for new and existing applications.
- **WCF uses cross-platform standards.** WCF was designed with cross-platform interoperability in mind and supports many industry standards (SOAP, WS-Security, WS-Trust, etc.). A WCF service can interoperate with clients running on operating systems other than Windows. Remoting was designed primarily for

environments where both the server and client applications run using the .NET framework on a Windows operating system.

- **WCF has built-in security.** WCF was designed with security in mind and offers many options for authentication, transport level security, message level security, etc. Remoting was designed to make it easy for applications to interoperate but was not designed to be secure in non-trusted environments. WCF was designed to work in both trusted and non-trusted environments.

Migration Recommendations

The following are the recommended steps to migrate from .NET Remoting to WCF:

- **Create the service contract.** Define your service interface types, and mark them with the [ServiceContract] attribute. Mark all the methods the clients will be allowed to call with [OperationContract].
- **Create the data contract.** Define the data types that will be exchanged between server and client, and mark them with the [DataContract] attribute. Mark all the fields and properties the client will be allowed to use with [DataMember].
- **Create the fault contract (optional).** Create the types that will be exchanged between server and client when errors are encountered. Mark these types with [DataContract] and [DataMember] to make them serializable. For all service operations you marked with [OperationContract], also mark them with [FaultContract] to indicate which errors they may return.
- **Configure and host the service.** Once the service contract has been created, the next step is to configure a binding to expose the service at an endpoint. For more information, see [Endpoints: Addresses, Bindings, and Contracts](#).

Once a Remoting application has been migrated to WCF, it is still important to remove dependencies on .NET Remoting. This ensures that any Remoting vulnerabilities are removed from the application. These steps include the following:

- **Discontinue use of MarshalByRefObject.** The MarshalByRefObject type exists only for Remoting and is not used by WCF. Any application types that sub-class MarshalByRefObject should be removed or changed. The MarshalByRefObject type exists only for Remoting and is not used by WCF. Any application types that sub-class MarshalByRefObject should be removed or changed.
- **Discontinue use of [Serializable] and ISerializable.** The [Serializable] attribute and ISerializable interface were originally designed to serialize types within trusted environments, and they are used by Remoting. WCF serialization relies on types being marked with [DataContract] and [DataMember]. Data types used by an application should be modified to use [DataContract] and not to use ISerializable or [Serializable]. The [Serializable] attribute and ISerializable interface were originally designed to serialize types within trusted environments, and they are used by Remoting. WCF serialization relies on types being marked with [DataContract] and [DataMember]. Data types used by an application should be modified to use [DataContract] and not to use ISerializable or [Serializable].

Migration Scenarios

Now let's see how to accomplish the following common Remoting scenarios in WCF:

1. Server returns an object by-value to the client
2. Server returns an object by-reference to the client
3. Client sends an object by-value to the server

NOTE

Sending an object by-reference from the client to the server is not allowed in WCF.

When reading through these scenarios, assume our baseline interfaces for .NET Remoting look like the following example. The .NET Remoting implementation is not important here because we want to illustrate only how to use WCF to implement equivalent functionality.

```
public class RemotingServer : MarshalByRefObject
{
    // Demonstrates server returning object by-value
    public Customer GetCustomer(int customerId) {...}

    // Demonstrates server returning object by-reference
    public CustomerReference GetCustomerReference(int customerId) {...}

    // Demonstrates client passing object to server by-value
    public bool UpdateCustomer(Customer customer) {...}
}
```

Scenario 1: Service Returns an Object by Value

This scenario demonstrates a server returning an object to the client by value. WCF always returns objects from the server by value, so the following steps simply describe how to build a normal WCF service.

1. Start by defining a public interface for the WCF service and mark it with the [ServiceContract] attribute. We use [OperationContract] to identify the server-side methods our client will call.

```
[ServiceContract]
public interface ICustomerService
{
    [OperationContract]
    Customer GetCustomer(int customerId);

    [OperationContract]
    bool UpdateCustomer(Customer customer);
}
```

2. The next step is to create the data contract for this service. We do this by creating classes (not interfaces) marked with the [DataContract] attribute. The individual properties or fields we want visible to both client and server are marked with [DataMember]. If we want derived types to be allowed, we must use the [KnownType] attribute to identify them. The only types WCF will allow to be serialized or deserialized for this service are those in the service interface and these "known types". Attempting to exchange any other type not in this list will be rejected.

```

[DataContract]
[KnownType(typeof(PremiumCustomer))]
public class Customer
{
    [DataMember]
    public string FirstName { get; set; }

    [DataMember]
    public string LastName { get; set; }

    [DataMember]
    public int CustomerId { get; set; }
}

[DataContract]
public class PremiumCustomer : Customer
{
    [DataMember]
    public int AccountId { get; set; }
}

```

3. Next, we provide the implementation for the service interface.

```

public class CustomerService : ICustomerService
{
    public Customer GetCustomer(int customerId)
    {
        // read from database
    }

    public bool UpdateCustomer(Customer customer)
    {
        // write to database
    }
}

```

4. To run the WCF service, we need to declare an endpoint that exposes that service interface at a specific URL using a specific WCF binding. This is typically done by adding the following sections to the server project's web.config file.

```

<configuration>
  <system.serviceModel>
    <services>
      <service name="Server.CustomerService">
        <endpoint address="http://localhost:8083/CustomerService"
          binding="basicHttpBinding"
          contract="Shared.ICustomerService" />
      </service>
    </services>
  </system.serviceModel>
</configuration>

```

5. The WCF service can then be started with the following code:

```

ServiceHost customerServiceHost = new ServiceHost(typeof(CustomerService));
customerServiceHost.Open();

```


When this ServiceHost is started, it uses the web.config file to establish the proper contract, binding and endpoint. For more information about configuration files, see [Configuring Services Using Configuration Files](./configuring-services-using-configuration-files.md). This style of starting the server is known as self-hosting. To learn more about other choices for hosting WCF services, see [Hosting Services](./hosting-services.md).

6. The client project's app.config must declare matching binding information for the service's endpoint. The easiest way to do this in Visual Studio is to use **Add Service Reference**, which will automatically update the app.config file. Alternatively, these same changes can be added manually.

```
<configuration>
  <system.serviceModel>
    <client>
      <endpoint name="customerservice"
        address="http://localhost:8083/CustomerService"
        binding="basicHttpBinding"
        contract="Shared.ICustomerService"/>
    </client>
  </system.serviceModel>
</configuration>
```

For more information about using **Add Service Reference**, see [How to: Add, Update, or Remove a Service Reference](#).

7. Now we can call the WCF service from the client. We do this by creating a channel factory for that service, asking it for a channel, and directly calling the method we want on that channel. We can do this because the channel implements the service's interface and handles the underlying request/reply logic for us. The return value from that method call is the deserialized copy of the server's response.

```
ChannelFactory<ICustomerService> factory =
    new ChannelFactory<ICustomerService>("customerservice");
ICustomerService service = factory.CreateChannel();
Customer customer = service.GetCustomer(42);
Console.WriteLine(String.Format("  Customer {0} {1} received.",
    customer.FirstName, customer.LastName));
```

Objects returned by WCF from the server to the client are always by value. The objects are deserialized copies of the data sent by the server. The client can call methods on these local copies without any danger of invoking server code through callbacks.

Scenario 2: Server Returns an Object By Reference

This scenario demonstrates the server providing an object to the client by reference. In .NET Remoting, this is handled automatically for any type derived from `MarshalByRefObject`, which is serialized by-reference. An example of this scenario is allowing multiple clients to have independent sessionful server-side objects. As previously mentioned, objects returned by a WCF service are always by value, so there is no direct equivalent of a by-reference object, but it is possible to achieve something similar to by-reference semantics using an [EndpointAddress10](#) object. This is a serializable by-value object that can be used by the client to obtain a sessionful by-reference object on the server. This enables the scenario of having multiple clients with independent sessionful server-side objects.

1. First, we need to define a WCF service contract that corresponds to the sessionful object itself.

```
[ServiceContract(SessionMode = SessionMode.Allowed)]
public interface ISessionBoundObject
{
    [OperationContract]
    string GetCurrentValue();

    [OperationContract]
    void SetCurrentValue(string value);
}
```

> [!TIP]
> Notice that the sessionful object is marked with [ServiceContract], making it a normal WCF service interface. Setting the SessionMode property indicates it will be a sessionful service. In WCF, a session is a way of correlating multiple messages sent between two endpoints. This means that once a client obtains a connection to this service, a session will be established between the client and the server. The client will use a single unique instance of the server-side object for all its interactions within this single session.

2. Next, we need to provide the implementation of this service interface. By denoting it with [ServiceBehavior] and setting the InstanceContextMode, we tell WCF we want to use a unique instance of this type for an each session.

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)]
public class MySessionBoundObject : ISessionBoundObject
{
    private string _value;

    public string GetCurrentValue()
    {
        return _value;
    }

    public void SetCurrentValue(string val)
    {
        _value = val;
    }
}
```

3. Now we need a way to obtain an instance of this sessionful object. We do this by creating another WCF service interface that returns an EndpointAddress10 object. This is a serializable form of an endpoint that the client can use to create the sessionful object.

```
[ServiceContract]
public interface ISessionBoundFactory
{
    [OperationContract]
    EndpointAddress10 GetInstanceAddress();
}
```

And we implement this WCF service:

```

public class SessionBoundFactory : ISessionBoundFactory
{
    public static ChannelFactory<ISessionBoundObject> _factory =
        new ChannelFactory<ISessionBoundObject>("sessionbound");

    public SessionBoundFactory()
    {
    }

    public EndpointAddress10 GetInstanceAddress()
    {
        IClientChannel channel = (IClientChannel)_factory.CreateChannel();
        return EndpointAddress10.FromEndpointAddress(channel.RemoteAddress);
    }
}

```

This implementation maintains a singleton channel factory to create sessionful objects. When `GetInstanceAddress()` is called, it creates a channel and creates an `EndpointAddress10` object that effectively points to the remote address associated with this channel. `EndpointAddress10` is simply a data type that can be returned to the client by-value.

4. We need to modify the server's configuration file by doing the following two things as shown in the example below:
 - a. Declare a `<client>` section that describes the endpoint for the sessionful object. This is necessary because the server also acts as a client in this situation.
 - b. Declare endpoints for the factory and sessionful object. This is necessary to allow the client to communicate with the service endpoints to acquire the `EndpointAddress10` and to create the sessionful channel.

```

<configuration>
  <system.serviceModel>
    <client>
      <endpoint name="sessionbound"
        address="net.tcp://localhost:8081/SessionBoundObject"
        binding="netTcpBinding"
        contract="Shared.ISessionBoundObject"/>
    </client>
    <services>
      <service name="Server.CustomerService">
        <endpoint address="http://localhost:8083/CustomerService"
          binding="basicHttpBinding"
          contract="Shared.ICustomerService" />
      </service>
      <service name="Server.MySessionBoundObject">
        <endpoint address="net.tcp://localhost:8081/SessionBoundObject"
          binding="netTcpBinding"
          contract="Shared.ISessionBoundObject" />
      </service>
      <service name="Server.SessionBoundFactory">
        <endpoint address="net.tcp://localhost:8081/SessionBoundFactory"
          binding="netTcpBinding"
          contract="Shared.ISessionBoundFactory" />
      </service>
    </services>
  </system.serviceModel>
</configuration>

```

And then we can start these services:

```

ServiceHost factoryHost = new ServiceHost(typeof(SessionBoundFactory));
factoryHost.Open();

ServiceHost sessionHost = new ServiceHost(typeof(MySessionBoundObject));
sessionHost.Open();

```

5. We configure the client by declaring these same endpoints in its app.config file.

```

<configuration>
  <system.serviceModel>
    <client>
      <endpoint name="customerservice"
        address="http://localhost:8083/CustomerService"
        binding="basicHttpBinding"
        contract="Shared.ICustomerService"/>
      <endpoint name="sessionbound"
        address="net.tcp://localhost:8081/SessionBoundObject"
        binding="netTcpBinding"
        contract="Shared.ISessionBoundObject"/>
      <endpoint name="factory"
        address="net.tcp://localhost:8081/SessionBoundFactory"
        binding="netTcpBinding"
        contract="Shared.ISessionBoundFactory"/>
    </client>
  </system.serviceModel>
</configuration>

```

6. In order to create and use this sessionful object, the client must do the following steps:

- a. Create a channel to the ISessionBoundFactory service.
- b. Use that channel to invoke that service to obtain an EndpointAddress10.
- c. Use the EndpointAddress10 to create a channel to obtain a sessionful object.
- d. Interact with the sessionful object to demonstrate it remains the same instance across multiple calls.

```

ChannelFactory<ISessionBoundFactory> channelFactory =
    new ChannelFactory<ISessionBoundFactory>("factory");
ISessionBoundFactory sessionFactory = channelFactory.CreateChannel();

EndpointAddress10 address1 = sessionFactory.GetInstanceAddress();
EndpointAddress10 address2 = sessionFactory.GetInstanceAddress();

ChannelFactory<ISessionBoundObject> sessionObjectFactory1 =
    new ChannelFactory<ISessionBoundObject>(new NetTcpBinding(),
        address1.ToEndpointAddress());
ChannelFactory<ISessionBoundObject> sessionObjectFactory2 =
    new ChannelFactory<ISessionBoundObject>(new NetTcpBinding(),
        address2.ToEndpointAddress());

ISessionBoundObject sessionInstance1 = sessionObjectFactory1.CreateChannel();
ISessionBoundObject sessionInstance2 = sessionObjectFactory2.CreateChannel();

sessionInstance1.SetCurrentValue("Hello");
sessionInstance2.SetCurrentValue("World");

if (sessionInstance1.GetCurrentValue() == "Hello" &&
    sessionInstance2.GetCurrentValue() == "World")
{
    Console.WriteLine("sessionful server object works as expected");
}

```

WCF always returns objects by value, but it is possible to support the equivalent of by-reference semantics through the use of `EndpointAddress10`. This permits the client to request a sessionful WCF service instance, after which it can interact with it like any other WCF service.

Scenario 3: Client Sends Server a By-Value Instance

This scenario demonstrates the client sending a non-primitive object instance to the server by value. Because WCF only sends objects by value, this scenario demonstrates normal WCF usage.

1. Use the same WCF Service from Scenario 1.
2. Use the client to create a new by-value object (`Customer`), create a channel to communicate with the `ICustomerService` service, and send the object to it.

```
ChannelFactory<ICustomerService> factory =  
    new ChannelFactory<ICustomerService>("customerservice");  
ICustomerService service = factory.CreateChannel();  
PremiumCustomer customer = new PremiumCustomer {  
    FirstName = "Bob",  
    LastName = "Jones",  
    CustomerId = 43,  
    AccountId = 99};  
bool success = service.UpdateCustomer(customer);  
Console.WriteLine(String.Format("  Server returned {0}.", success));
```

The customer object will be serialized, and sent to the server, where it is deserialized into a new copy of that object.

> [!NOTE]
> This code also illustrates sending a derived type (`PremiumCustomer`). The service interface expects a `Customer` object, but the `[KnownType]` attribute on the `Customer` class indicated `PremiumCustomer` was also allowed. WCF will fail any attempt to serialize or deserialize any other type through this service interface.

Normal WCF exchanges of data are by value. This guarantees that invoking methods on one of these data objects executes only locally – it will not invoke code on the other tier. While it is possible to achieve something like by-reference objects returned *from* the server, it is not possible for a client to pass a by-reference object *to* the server. A scenario that requires a conversation back and forth between client and server can be achieved in WCF using a duplex service. For more information, see [Duplex Services](#).

Summary

.NET Remoting is a communication framework intended to be used only within fully-trusted environments. It is a legacy product and supported only for backward compatibility. It should not be used to build new applications. Conversely, WCF was designed with security in mind and is recommended for new and existing applications. Microsoft recommends that existing Remoting applications be migrated to use WCF or ASP.NET Web API instead.

Using the WCF Development Tools

8/31/2018 • 2 minutes to read • [Edit Online](#)

This section describes the Visual Studio development tools that can assist you in developing your WCF service.

You can use the Visual Studio templates as a foundation to quickly build your own service, then use WCF Service Auto Host and WCF Test Client to debug and test your service. These tools together provide a fast and seamless debug and testing cycle, and preclude the need to commit to a hosting model at an early stage.

The WCF Developer Tools

[WCF Visual Studio Templates](#)

You can use the predefined Visual Studio project and item templates in Visual Studio to quickly build WCF services and surrounding applications.

[WCF Service Host \(WcfSvcHost.exe\)](#)

The WCF Service Auto Host (WcfSvcHost.exe) allows you to launch the Visual Studio debugger (F5) to automatically host and test a service you have implemented. You can then test the service using the WCF Test Client (wcfTestClient.exe) or your own client to find and fix any potential errors.

[WCF Test Client \(WcfTestClient.exe\)](#)

WCF Test Client (WcfTestClient.exe) is a GUI tool that allows you to input parameters of arbitrary types, submit that input to the service, and view the response the service sends back. It provides a seamless service testing experience when combined with WCF Service Auto Host.

[Generating Data Type Classes from XML](#)

XML data stored in the clipboard can be pasted into a code page. The classes defined in the data will be converted to code types.

Using the Tools without Administrator privilege

To enable users without administrator privilege to develop WCF services, an ACL (Access Control List) is created for the namespace "http://+:8731/Design_Time_Addresses" during the installation of Visual Studio. The ACL is set to (UI), which includes all interactive users logged on to the machine. Administrators can add or remove users from this ACL, or open additional ports. This ACL enables WCF or WF templates to send and receive data in their default configuration. It also enables users to use the WCF Service Auto Host (wcfSvcHost.exe) without granting them administrator privileges.

You can modify access using the Netsh.exe tool in Windows Vista under the elevated administrator account. The following is an example of using Netsh.exe.

```
netsh http add urlacl url=http://+:8001/MyService user=<domain>\<user>
```

For more information about Netsh.exe, see [How to Use the Netsh.exe Tool and Command-Line Switches](#).

See Also

[WCF Visual Studio Templates](#)

[WCF Service Host \(WcfSvcHost.exe\)](#)

WCF Service Host (WcfSvcHost.exe)

8/31/2018 • 6 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) Service Host (WcfSvcHost.exe) allows you to launch the Visual Studio debugger (F5) to automatically host and test a service you have implemented. You can then test the service using WCF Test Client (WcfTestClient.exe), or your own client, to find and fix any potential errors.

WCF Service Host

WCF Service Host enumerates the services in a WCF service project, loads the project's configuration, and instantiates a host for each service that it finds. The tool is integrated into Visual Studio through the WCF Service template and is invoked when you start to debug your project.

By using WCF Service Host, you can host a WCF service (in a WCF service library project) without writing extra code or committing to a specific host during development.

NOTE

WCF Service Host does not support Partial Trust. If you want to use a WCF Service in Partial Trust, do not use the WCF Service Library Project template in Visual Studio to build your service. Instead, create a New WebSite in Visual Studio by choosing the WCF Service WebSite template, which can host the service in a WebServer on which WCF Partial Trust is supported.

Project Types hosted by WCF Service Host

WCF Service Host can host the following WCF service library project types: WCF Service Library, Sequential Workflow Service Library, State Machine Workflow Service Library and Syndication Service Library. WCF Service Host can also host those services that can be added to a service library project using the **Add Item** functionality. This includes WCF Service, WF State Machine Service, WF Sequential Service, XAML WF State Machine Service and XAML WF Sequential Service.

You should note, however, that the tool will not help you to configure a host. For this task, you must manually edit the App.config file. The tool also does not validate user-defined configuration files.

Caution

You should not use WCF Service Host to host services in a production environment, as it was not engineered for this purpose. WCF Service Host does not support the reliability, security, and manageability requirements of such an environment. Instead, use IIS since it provides superior reliability and monitoring features, and is the preferred solution for hosting services. Once development of your services is complete, you should migrate the services from WCF Service Host to IIS.

Scenarios for Using WCF Service Host inside Visual Studio

The following table lists all the parameters in the **Command line arguments** dialog box, which can be found by right-clicking your project in **Solutions Explorer** in Visual Studio, selecting **Properties**, then selecting the **Debug** tab and clicking **Start Project**. These parameters are useful in configuring WCF Service Host.

PARAMETER	MEANING
-----------	---------

PARAMETER	MEANING
<code>/client</code>	An optional parameter that specifies the path to an executable to run after the services are hosted. This launches WCF Test Client following hosting.
<code>/clientArg</code>	Specify a string as an argument that is passed to the custom client application.
<code>/?</code>	Displays the help text.

Using WCF Test Client

After you create a new WCF service project and press F5 to start the debugger, WCF Service Host starts hosting all the services it finds in your project. WCF Test Client automatically opens and displays a list of service endpoints defined in the configuration file. From the main window, you can test the parameters and invoke your service.

To make sure that WCF Test Client is used, right-click your project in **Solutions Explorer** in Visual Studio, select **Properties**, then select the **Debug** tab. Click **Start Project** and ensure that the following appears in the **Command line arguments** dialog box.

```
/client:WcfTestClient.exe
```

Using a Custom Client

To use a custom client, right-click your project in **Solutions Explorer** in Visual Studio, select **Properties**, then select the **Debug** tab. Click **Start Project** and edit the `/client` parameter in the **Command line arguments** dialog box to point to your custom client, as indicated in the following example.

```
/client:"path/CustomClient.exe"
```

When you press F5 to start the service again, WCF Service Host automatically starts your custom client when you launch the debugger.

You can also use the `/clientArg:` parameter to specify a string as an argument that is passed to the custom client application, as indicated in the following example.

```
/client:"path/CustomClient.exe" /clientArg:"arguments that are passed to Client"
```

For example, if you are using the Syndication Service Library template, you can use the following command line arguments,

```
/client:iexplore.exe /clientArgs:http://localhost:8731/Design_Time_Addresses/Feed1/
```

Specifying No Client

To specify that no client will be used after WCF service hosting, right-click your project in **Solutions Explorer** in Visual Studio, select **Properties**, then select the **Debug** tab. Click **Start Project** and leave the **Command line arguments** dialog box blank.

Using a Custom Host

To use a custom host, right-click your project in **Solutions Explorer** in Visual Studio, select **Properties**, then select the **Debug** tab. Click **Start External Program** and enter the full path to the custom host. You can also use the **Command line arguments** dialog box to specify arguments to be passed to the host.

WCF Service Host User Interface

When you initially invoke WCF Service Host (by pressing F5 inside Visual Studio), the **WCF Service Host** window automatically opens. When WCF Service Host is running, the program's icon appears in the notification area. Double-click the icon to open the **WCF Service Host** window

When errors occur during service hosting, WCF Service Host dialog box will open to display relevant information.

The **WCF Service Host** main window contains two menus:

- **File:** Contains the **Close** and **Exit** commands. When you click **Close**, the **WCF Service Host** dialog box closes, but the services continue to be hosted. When you click **Exit**, WCF Service Host is also shut down. This also stops all hosted services.
- **Help:** Contains the **About** command that contains version information. It also contains the **Help** command that can open a help file.

The main **WCF Service Host** window contains two areas:

- The first area is **Service**. It contains a list that displays basic information of all services. The information includes:
 - **Service:** Lists all the services.
 - **Status:** Lists the status of the service. Valid values are "Started", "Stopped," and "Error".
 - **Metadata Address:** Displays the metadata address of the services.
- The second area is **Additional Information**. It displays a detailed explanation of the service status when the specific service line is selected in the **Service** area. If the status is Error, you can view the full error message on the screen.

Stopping WCF Service Host

You can shut down WCF Service Host in the following four ways:

- Stop the debugging session in Visual Studio.
- Select **Exit** from the **File** menu in the **WCF Service Host** window.
- Select **Exit** from context menu of WCF Service Host tray icon in the system notification area.
- Exit WCF Test Client if it is being used.

Using Service Host without Administrator privilege

To enable users without administrator privilege to develop WCF services, an ACL (Access Control List) is created for the namespace "http://+:8731/Design_Time_Addresses" during the installation of Visual Studio. The ACL is set to (UI), which includes all interactive users logged on to the machine. Administrators can add or remove users from this ACL, or open additional ports. This ACL enables users to use the WCF Service Auto Host (wcfSvcHost.exe) without granting them administrator privileges.

You can modify access using the netsh.exe tool in Windows Vista under the elevated administrator account. The following is an example of using netsh.exe.

```
netsh http add urlacl url=http://+:8001/MyService user=<domain>\<user>
```

For more information on netsh.exe, see ["How to Use the Netsh.exe Tool and Command-Line Switches"](#).

See Also

[WCF Test Client \(WcfTestClient.exe\)](#)

WCF Test Client (WcfTestClient.exe)

10/25/2018 • 9 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) Test Client (WcfTestClient.exe) is a GUI tool that enables users to input test parameters, submit that input to the service, and view the response that the service sends back. It provides a seamless service testing experience when combined with WCF Service Host.

You can typically find the WCF Test Client (WcfTestClient.exe) in the following location:

`C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\Common7\IDE` - Community may be one of "Enterprise", "Professional" or "Community" depending on which level of Visual Studio is installed.

Scenarios for Using Test Client

The following sections discuss the most common scenarios in which you can use WCF Test Client to streamline your development process.

Inside Visual Studio

WCF Service Host Starts WCF Test Client with a Single Service

After you create a new WCF service project and press F5 to start the debugger, the WCF Service Host begins to host the service in your project. Then, WCF Test Client opens and displays a list of service endpoints defined in the configuration file. You can test the parameters and invoke the service, and repeat this process to continuously test and validate your service.

WCF Service Host Starts WCF Test Client with Multiple Services

You can also use WCF Test Client to help debug a service project that contains multiple services. When WCF Test Client opens, it automatically iterates the list of services in your project and opens them for testing.

Outside Visual Studio

You can also invoke the WCF Test Client (WcfTestClient.exe) outside Visual Studio to test an arbitrary service on the Internet. To locate the tool, go to the following location:

`C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\Common7\IDE` (where community can be one of "Enterprise", "Professional" or "Community" depending on which level of Visual Studio is installed on the machine)

To use the tool, double-click the file name to open it from this location, or launch it from a command line.

WCF Test Client takes an arbitrary number of URIs as command line arguments. These are the URIs of services that can be tested.

```
wcfTestClient.exe URI1 URI2 ...
```

After the WCF Test Client window is opened, click **File->Add Service**, and enter the endpoint address of the service you want to open.

WCF Test Client User Interface

You can use WCF Test Client with a single service or multiple services.

Service Operations

The left pane of the WCF Test Client main window lists all the available services, along with their respective endpoints and operations.

When you double-click an operation, you can view its content in the right pane inside a new tab with the operation's name.

The left pane also lists client configuration files. Double-click any of the items to display the content of the file in a new tabbed window in the right pane.

Entering Test Parameters

To view the test parameters, double-click an operation to open it in the right pane. The parameters are showed in **Formatted** view by default, and you can enter arbitrary values for the parameters to test the service.

To view the message's XML, click **XML**. To send them to the service, click **Invoke**.

For a DataSet parameter, click the ... button next to **Edit...** to edit it in a new window showing the DataGrid. Notice the appearance of the **Copy DataSet** and **Paste DataSet** buttons. If the schema of the DataSet object is unknown upon the first edit, the DataGrid is empty. You have to paste a DataSet object with the same schema into the current object in the DataGrid. (Notice that you need to copy the schema from somewhere else before the paste operation.) You can also copy a Dataset object for future usage by clicking the **Copy DataSet** button.

The service's response appears below the test parameters.

NOTE

If the expected return value is a string, the result will be displayed as a quoted string even though the input provided was not in quotes.

If you specified a particular operation as one-way when you created the contract for the service, no service response is displayed. As soon as the message is queued for delivery, a dialog box pops up to notify you that the message was successfully sent.

Session Support

The **Start a new proxy** check box in a service operation's tab enables you to toggle session support. This box is cleared by default.

When you enter test parameters for a specific operation (or another operation in the same service endpoint) and click **Invoke** multiple times with the check box cleared, these operations share one proxy and the service status is persisted across multiple operations.

If the **Start a new proxy** check box is checked, a new proxy is started for each **Invoke**, the previous session scenario is ended, and the service status is reset.

Editing Client Configuration

The left pane of the WCF Test Client main window lists client configuration files. Double-click any of the items to display the contents of the file in the right pane.

Edit with Service Configuration Editor

Right-click **Config File** in the left pane and select the context menu **Edit with SvcConfigEditor**. Service Configuration Editor is launched with the client configuration content. You can edit the configuration and save it within the tool.

After saving the file in Service Configuration Editor, WCF Test Client displays a warning message to inform you that the file has been modified outside and asks whether you would like to reload it.

If you select **Yes**, the configuration content in the "Client.dll.config" tab reflects the changes you made in the editor.

If you select **No**, the configuration content in the "Client.dll.config" tab remains unchanged and the modified content is automatically saved to the source file.

Restore to Default Configuration

If you want to cancel all the changes and restore to the default client configuration, right-click **Config File** in the left pane and select the context menu **Restore to Default Config**. The default configuration value is loaded and content in "Client.dll.config" tab is restored.

Validate Changes

When saved changes are being loaded in WCF Test Client, the configuration is checked for validity against WCF schema. If errors are found, a dialog box is displayed to show error details.

During proxy generation, binary compiling, or service invoking, menu items that support editing (that is, "Edit ...", "Restore ...", and so on) are disabled. Service invocation is also disabled when loading updated configuration to WCF Test Client.

Persist Client Configuration

The **Tools->Options->Client Configuration** tab contains an **Always Regenerate Config When Launching Services** option, which is enabled by default. This option specifies that every time WCF Test Client loads a service, it regenerates a configuration file based on the latest service contract and service App.config files.

If you have edited the client configuration for your WCF service and want to always use this updated file to debug your service, you can uncheck the **Regenerate** option. By doing so, even when you update the service and reopen WCF Test Client, the Client.dll.config file is the one you updated previously instead of a regenerated one based on the updated service.

However, you might need to edit the configuration file to make it consistent with the regenerated proxy. If the regenerated proxy and configuration file are mismatched due to an updated service, errors will occur when the service is invoked.

Caution

If you have modified the client configuration file and select to reuse it in the future, you can find the file in the following location:

\Documents and Settings\[User Account]\My Documents\Test Client Projects.

Any updated credential information stored to the client configuration file is protected by the Access Control List (ACL) of this folder.

Adding, Removing and Refreshing Services

Add Service

Click **File->Add Service** to add a service to WCF Test Client. You are then required to type the URI (endpoint address) of the service to be added. The service's address can be a mex address or WSDL address.

You can also find a list of 10 recently added services' endpoints in the **Recent Services** submenu. If you select one of them, the specified service is added to WCF Test Client.

You can also right-click the root of service tree **My Service Projects**, and select **Add Service** to achieve the same result.

During proxy generation, binary compiling, or service invocation, menu items that support adding a service are disabled. Service invocation is also disabled.

Remove Service

Right-click the service root of the service to be removed, and select **Remove Service** to remove a service from WCF Test Client.

During proxy generation, binary compiling, or service invocation, menu items that support removing a service are disabled. Service invocation is also disabled.

Refresh Service

If a change is made to the service while WCF Test Client is running and you want to ensure that the WCF Test Client implementation for that service is up-to-date, right-click the service root of the service, and select **Refresh**

Service. Note that after refreshing, the service status is reset.

During proxy generation, binary compiling, or service invocation, menu items that support refreshing a service are disabled. Service invocation is also disabled.

Location of Files Generated by the Test Client

By default, WCF Test Client stores generated client code and configuration files in the "%appdata%\Local\temp\Test Client Projects" folder. This folder is deleted after WCF Test Client exits. If a configuration file is modified in WCF Test Client and the **Always Regenerate Config When Launching Services** option is disabled, the modified file is copied to the "Cached Config" folder under "My Documents\Test Client Projects Documents\Test Client Projects" with a mapping (metadata-address-to-file-name) XML file as an index.

You can also start WCF Test Client in a command line, use the `/ProjectPath` switch to specify a new desired path for storing generated files, or use the `/RestoreProjectPath` switch to restore the default location. The syntax is as follows:

```
wcfTestClient.exe /ProjectPath [desired location]
```

Running this command does not open WCF Test Client. Only the folder location is changed. You can run this command whether WCF Test Client is running or not. The new location is applied when WCF Test Client is restarted. The location information can be saved in registry, or in the WcfTestClient.exe.option file in the "%appdata%\Local\temp\Test Client Projects" folder.

Features supported by WCF Test Client

The following is a list of features supported by WCF Test Client:

- Service Invocation: Request/Response and One-way message.
- Bindings: all bindings supported by Svcutil.exe.
- Controlling Session.
- Message Contract.
- XML serialization.

The following is a list of features not supported by WCF Test Client:

- Types: [Stream](#), [Message](#), [XmlElement](#), [XmlAttribute](#), [XmlNode](#), types that implement the [IXmlSerializable](#) interface, including the related [XmlSchemaProviderAttribute](#) attribute, and the [XDocument](#) and [XElement](#) types and the ADO.NET [DataTable](#) type.
- Duplex contract.
- Transaction.
- Security: CardSpace, Certificate, and Username/Password.
- Bindings: WSFederationbinding, any Context bindings and Https binding, WebHttpbinding (Json response message support).

Closing WCF Test Client

You can close WCF Test Client in the following ways:

- On the **File** menu, click **Exit**. Alternatively, in the WCF Test Client main window, click **Close**. Both of these

actions also shut down WCF Service Auto Host and stop the Visual Studio debugging process if WCF Test Client was launched by Visual Studio.

- Right-click the **WCF Service Host** icon in the notification area, and then click **Exit**. This shuts down both WCF Service Auto Host and WCF Test Client and stops the Visual Studio debugging process.

See Also

[WCF Service Host \(WcfSvcHost.exe\)](#)

WCF Visual Studio Templates

8/31/2018 • 4 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) Visual Studio templates are predefined project and item templates you can use in Visual Studio to quickly build WCF services and surrounding applications.

Using the WCF Templates

WCF Visual Studio templates provide a basic class structure for service development. Specifically, these templates provide the basic definitions for service contract, data contract, service implementation, and configuration. You can use these templates to create a simple service with minimal code interaction, as well as a building block for more advanced services.

WCF Service Library Project Template

The WCF Service Library project template is available in the new project dialog box under **Visual C#\WCF** and **Visual Basic\WCF**.

When you create a new project using the **WCF Service** template, the new project automatically includes the following three files:

- Service contract file (IService1.cs or IService1.vb). The service contract file is an interface that has WCF service attributes applied. This file provides a definition of a simple service to show you how to define your services, and includes parameter-based operations and a simple data contract sample. This is the default file displayed in the code editor after creating a WCF service project.
- Service implementation file (Service1.cs or Service1.vb). The service implementation file implements the contract defined in the service contract file.
- Application configuration file (App.config). The configuration file provides the basic elements of a WCF service model with a secure HTTP binding. It also includes an endpoint for the service and enables metadata exchange.

NOTE

Visual Studio is configured to recognize the App.config file as the configuration file for the project when it is run using the [WCF Service Host \(WcfSvcHost.exe\)](#), which is the default configuration. If you host the service library in an executable, you have to move the configuration code to the configuration file of the executable, as configuration files for DLLs are not valid.

WCF Service Application Template

The WCF Service Application template is available in the New Project dialog box under **Visual C#\WCF** and **Visual Basic\WCF**.

When you create a new project using the **WCF Web Application Service** template, the project includes the following four files:

- Service host file (service1.svc).
- Service contract file (IService1.cs or IService1.vb).
- Service implementation file (Service1.svc.cs or Service1.svc.vb).
- Web configuration file (Web.config).

The template automatically creates a Web site (to be deployed to a virtual directory) and hosts a service in it.

WCF Web Site Template

The WCF Web Site template is available in the New Project dialog box under **Visual C#\Web Site\WCF Service** and **Visual Basic\Web Site\WCF Service**. This creates the same files as the WCF Service Application template but organizes it as if it were a ASP.NET web site. App_Code and App_Data folders are created.

WCF Service Item Template

The WCF Service Item template is a custom template that provides a quick way to add WCF services to your existing Visual Studio projects.

To use this template, go to the **Solution Explorer** pane, right-click your project name, point to **Add**, and then click **New Item** to launch the **Add New Item** dialog box.

The service interface and implementation files are placed in the root project folder.

The template attempts to merge the configuration section of the new service to the existing configuration file, if they are compatible types.

A service host file (service1.svc) is also created if the existing project is a Web project.

WCF WF Service Project and Item Template.

These templates create WCF services that host a Workflow Service, which is a workflow that can be accessed like a web service. Separate templates exist for XAML or imperative programming models. Using the templates, you can create sequential or state machine workflow. For more information on these types of workflow, see [Windows Workflow Foundation Tutorials](#). For more information about creating workflow projects, see [Creating Legacy Workflow Projects](#).

Visual Studio designer is more responsive when XOML type workflows are used instead of code based ones. XOML workflow is the default workflow type to be created.

WCF Syndication Service Library Template

This template enables you to expose your feed in the RSS or ATOM format as a WCF service. For more information, see [WCF Syndication](#).

Changing the Address of the Feed

The syndication template uses Internet Explorer during execution. When you right-click your project in **Solutions Explorer** in Visual Studio, select **Properties**, then select the **Debug** tab and you can see the default address of the template. Internet Explorer attempts to open the feed at this address.

If you change the address of your feed, you must also change the address in the **Debug** tab. If you do not do this, Internet Explorer attempts to open the feed at the default address and fail.

AJAX enabled WCF Service Item Template

This template exposes an AJAX control as a WCF service. For more information on AJAX controls, see the [AJAX control documentation](#).

Silverlight-enabled WCF Service Item Template

This template creates a Web service that provides data to a Silverlight client or front-end. The template can be added to a Web site or Web application project to create a WCF service, which includes service code and configuration that support communicating with a Silverlight client. You can then use **Add Service Reference** to add a client proxy of the service to the client, and exchange data between the Silverlight client and the Silverlight-enabled WCF service.

To access this template, right-click a Web site or Web application project in **Solution Explorer**, click **Add a new item**, and click **Silverlight-enabled WCF Service**.

NOTE

The Silverlight-enabled WCF Service exposes a `basicHttpBinding` endpoint without enabling any security settings. Therefore, information about the service can be obtained by all clients that connect to this service. Messages exchanged between the service and the client are also not signed or encrypted. To secure the endpoint properly, you should use ASP.NET authentication, HTTPS or other mechanisms.

See Also

[WCF Service Host \(WcfSvcHost.exe\)](#)

[WCF Test Client \(WcfTestClient.exe\)](#)

WCF Service Publishing

10/5/2018 • 4 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) Service Publishing assists you in progressing from the early development environment provided by WCF Service Host and WCF Test Client to actually deploying the application to a production environment for testing purposes. Before you commit to a final deployment plan, you can use Windows Communication Foundation (WCF) Service Publishing to verify that your WCF service performs correctly and is ready to be published. You can also choose to deploy your WCF service libraries to various target locations for testing.

Supported Services and Target Locations

WCF Service Publishing supports publishing WCF services created from the set of WCF service library templates, and their corresponding item templates, which include the following:

- WCF Service Library template with item template.
- Syndication Service Library.

You can find these service templates by choosing **File > New Project > [Visual Basic or Visual C#] > WCF**. For other WCF templates in this location (including WCF Workflow Service Application and WCF Service Application), you can publish using [One-Click publishing for web applications](#).

The service can be published to the following target locations.

- Local IIS.
- File System.
- FTP Site.

Using WCF Service Publishing

Perform the following steps to deploy a service implementation:

1. Open Visual Studio with elevated privileges (right-click the executable and choose **Run as administrator** to open it). If you are using IIS 7.0 or later, ensure that you have installed the "IIS Metabase and IIS6 Configuration Compatibility" component using "Turn Windows features on or off" in Control Panel.
2. Open a service project, select **Build > Publish <Project Name>** from the main menu, or right-click the project in **Solution Explorer** and click **Publish**.
3. The **Publish** window appears. Click the button to specify the target location that the service should be deployed to. You can select to deploy the application to local IIS, File System, or FTP Site. If deploying the application to local IIS, you can select your website and create your web application under it, by clicking the **Create New Web Application** icon at the top right corner.
4. After you click **Publish** in the main window, Visual Studio deploys the application to the specified target location and copies the Web.config, .svc, and assembly files to the target directory. . The name of .svc will be "ProjectName.ServiceName.svc". After the service is published successfully, you can find a hotlink in the Visual Studio Output window, which looks similar to "Connecting to `http://localhost/WebApplicationFolderName...`". You can press CTRL and click the link to open a browser page inside Visual Studio to view the service directory structure.

If you cannot browse to the site, it may be because directory browser is not enabled in IIS. Please follow the tips in the "Things you can try" section to enable it. Alternatively, you can directly type

`http://localhost/WebApplicationFolderName/ProjectName.ServiceName.svc` to view your service page.

You can use **Publish** to specify if you want to copy the assembly, configuration, and .svc file for all services defined in the project to the target location, and overwrite existing files at the destination.

If you choose to deploy your application to local IIS, you may encounter errors related to IIS setup. Please ensure that IIS is installed properly. You can enter `http://localhost` in your browser's address bar and check whether the IIS default page displays. In some cases, the issues may also be caused by improper registration of ASP.NET or WCF in IIS. You can open the Developer Command Prompt for Visual Studio and run the command `aspnet_regiis.exe -ir` to fix ASP.NET registration issues, or run command `ServiceModelReg.exe -ia` to fix WCF registration issues.

Files Generated for Publishing

Before a WCF service library can be Web-hosted, the following files are generated by the tool: assembly files, Web.config file, and .svc file. All the files are copied to the target location. The service is then published.

Assembly files

When you publish a WCF service using this tool, the service is automatically built first and the assembly files are generated in the service project after building.

.SVC File

The publishing operation generates a *.svc file for each WCF service, whether the file exists or not, to ensure version validity. There are two different kinds of svc files: one for WCF Service Library and Syndication Service Library, and another one for Sequential and State Machine Workflow Service Library. The generated *.svc file is copied to the root folder in the target location.

Web.config File

Each time a service project is published to a specific target location, a Web.config file is created.

The generated Web.config file includes Web sections that are useful for Web hosting, and the content of App.config for the WCF service library with the following changes:

- The base address is excluded.
- Settings in the `<diagnostics>` element are excluded to preserve the tracing settings of the target platform.

Publishing WCF services with non-HTTP Bindings to IIS

If you are using IIS 7.0 or later, you can publish WCF services with non-HTTP bindings to IIS. You need to do some pre-configurations. For more information, please see the topics at [Hosting in Windows Process Activation Service](#).

Security

Publishing to local IIS requires administrator privilege, because IIS requires running in Administrator account. If a user without administrator privilege opens WCF Service Publishing, IIS is not available as a target location. Publishing to File System, or FTP Site works without administrator privilege.

See Also

- [WCF Visual Studio Templates](#)
- [WCF Service Host \(WcfSvcHost.exe\)](#)
- [WCF Test Client \(WcfTestClient.exe\)](#)

Renaming a WCF Service

5/4/2018 • 2 minutes to read • [Edit Online](#)

This topic describes how you can rename a Windows Communication Foundation (WCF) service.

Renaming a WCF Service

Perform the following steps to rename a service in a Windows Communication Foundation (WCF) template,

- Change the name of the class that implements the service.
- In the configuration file of the service, change the name of the service to the new name you have chosen, as indicated in the following example. The configuration file can be either app.config or web.config file depending on your hosting model.

```
<system.servicemodel>
  <services>
    <service name="WcfService.NewName">
    </service>
  </services>
</system.servicemodel>
```

- If your service is webhosted, it uses an *.svc file. Open the svc file and modify the name of your service as indicated in the following example. This step is not necessary for self-hosted applications, as there is no svc file.

```
<%@ ServiceHost Service="WcfService.NewName"%>
```

Renaming a WCF Service Contract

- Perform the following steps to rename the service contract,
- Change the name of the service contract.
- In the configuration file of the service, change the name of the service contract to the new name you have chosen, as indicated in the following example. The configuration file can be either app.config or web.config file depending on your hosting model.

```
<system.servicemodel>
  <services>
    <service>
      <endpoint contract="WcfService.NewContractName" />
    </service>
  </services>
</system.servicemodel>
```

Deploying a WCF Library Project

8/31/2018 • 2 minutes to read • [Edit Online](#)

This topic describes how you can deploy a Windows Communication Foundation (WCF) Service Library Project.

Deploying a WCF Service Library

A WCF service library is a dynamic-link library (DLL). As such, it cannot be executed on its own. It needs to be deployed into a hosting environment. For more information about this process, see [Hosting and Consuming WCF Services](#).

A WCF service library can be deployed like any other WCF service. However, be aware that .NET Framework does not support configuration for DLLs. [System.Configuration](#) supports one configuration file per app-domain. The WCF service library project alleviates this limitation by providing an App.config file for the library during development. However, the App.config file is not recognized after deployment.

You have to move your configuration code into the configuration file recognized by your hosting environment. For self-hosting, you should copy the contents of the App.config file into the App.config file of the hosting executable. If you use IIS to host your service, you should copy the contents of the App.config file into the Web.config file of the virtual directory.

Controlling Auto-launching of WCF Service Host

5/5/2018 • 2 minutes to read • [Edit Online](#)

You can control the auto-launching capability of Windows Communication Foundation (WCF) Service Host (WcfSvcHost.exe) for a WCF Service Library project, when you debug another project in the same Visual Studio solution containing multiple projects.

To do so, right-click the WCF Service Project in **Solution Explorer**, choose **Properties**, and click **WCF Options** tab. The **Start WCF Service Host when debugging another project in the same solution** check box is enabled by default. You can clear the box so that WCF Service Host for this specific project is not launched when another project is debugged in the same solution.

This behavior does not affect the F5 debugging, or Add Service Reference functionalities for this project.

This option is available to the following projects:

- WCF Service Library Project.
- Sequential Workflow Service Library Project.
- State Machine Workflow Service Library Project.
- Syndication Service Library Project.

See Also

[WCF Service Host \(WcfSvcHost.exe\)](#)

Generating Data Type Classes from XML

5/4/2018 • 2 minutes to read • [Edit Online](#)

.NET Framework 4.5 includes a new feature to generate data type classes from XML. This topic describes how to automatically generate data types for the .NET Blog RSS feed.

Obtaining the XML from the .NET Blog RSS feed

1. In Internet Explorer, navigate to the [.NET Blog RSS feed](#).
2. Right-click the page and select **View Source**.
3. Copy the text of the feed by pressing **Ctrl+A** to select all text, and **Ctrl+C** to copy.

Creating the data types

1. Open a code file where the proxy is to be used. This file should be part of a .NET Framework 4.5 project.
2. Place the cursor in a location in the file outside any existing classes.
3. Select **Edit, Paste Special, Paste XML as Classes**.
4. Classes called `link`, `rss`, `rssChannel`, `rssChannelImage`, `rssChannelItem` and `rssChannelItemGuid` are created with the necessary members for accessing the elements in the RSS feed.

Using the generated classes

1. Once the classes are generated, they can be used in code like any other classes. The following code example returns a new instance of the `rssChannelImage` class.

```
var channelImage = new rssChannelImage()
{
    title = "MyImage",
    link = "http://www.contoso.com/images/channelImage.jpg",
    url = "http://www.contoso.com/entries/myEntry.html"
};
```

Windows Communication Foundation Tools

5/5/2018 • 2 minutes to read • [Edit Online](#)

Microsoft Windows Communication Foundation (WCF) tools are designed to make it easier for you to create, deploy, and manage WCF applications. This section contains detailed information about the tools. Please note that the tools are not supported.

You can run all the tools from the command line.

The following table lists these tools and provides a brief description.

TOOL	DESCRIPTION
ServiceModel Metadata Utility Tool (Svcutil.exe)	Generates service model code from metadata documents and metadata documents from service model code.
Find Private Key Tool (FindPrivateKey.exe)	Retrieves the private key from a specified store.
ServiceModel Registration Tool (ServiceModelReg.exe)	Manages the registration and un-registration of ServiceModel on a single machine.
COM+ Service Model Configuration Tool (ComSvcConfig.exe)	Configures COM+ interfaces to be exposed as Web services.
Configuration Editor Tool (SvcConfigEditor.exe)	Creates and modifies configuration settings for WCF services.
Service Trace Viewer Tool (SvcTraceViewer.exe)	Helps you view, group, and filter trace messages so that you can diagnose, repair, and verify issues with WCF services.
WS-AtomicTransaction Configuration Utility (wsatConfig.exe)	Configures basic WS-AtomicTransaction support settings using a command line tool.
WS-AtomicTransaction Configuration MMC Snap-in	Configures basic WS-AtomicTransaction support settings using a MMC snap-in.
WorkFlow Service Registration Tool (WFServicesReg.exe)	Registers a Windows Workflow service.
WCF Service Host (WcfSvcHost.exe)	Hosts WCF services contained in libraries (*.dll) files
WCF Test Client (WcfTestClient.exe)	A GUI tool that allows you to input parameters of arbitrary types, submit that input to the service, and view the response the service sends back.
Contract-First Tool	A Visual Studio build task that creates code classes from XSD data contracts.

All the preceding tools except ServiceModelReg.exe, WsatConfig.exe and ComSvcConfig.exe ship with the Windows SDK, and can be found under the C:\Program Files\Microsoft SDKs\Windows\v6.0\Bin folder. The specific 3 tools can be found under C:\Windows\Microsoft.NET\Framework\v3.0\Windows Communication Foundation.

ServiceModel Metadata Utility Tool (Svcutil.exe)

10/5/2018 • 15 minutes to read • [Edit Online](#)

The ServiceModel Metadata Utility tool is used to generate service model code from metadata documents, and metadata documents from service model code.

SvcUtil.exe

The ServiceModel Metadata Utility Tool can be found at the Windows SDK installation location, specifically *%ProgramFiles%\Microsoft SDKs\Windows\v6.0\Bin*.

Functionalities

The following table summarizes the various functionalities provided by this tool, and the corresponding topic that discusses how it is used:

TASK	TOPIC
Generates code from running services or static metadata documents.	Generating a WCF Client from Service Metadata
Exports metadata documents from compiled code.	How to: Use Svcutil.exe to Export Metadata from Compiled Service Code
Validates compiled service code.	How to: Use Svcutil.exe to Validate Compiled Service Code
Downloads metadata documents from running services.	How to: Use Svcutil.exe to Download Metadata Documents
Generates serialization code.	How to: Improve the Startup Time of WCF Client Applications using the XmlSerializer

Caution

Svcutil overwrites existing files on a disk if the names supplied as parameters are identical. This can include code files, configuration, or metadata files. To avoid this when generating code and configuration files, use the `/mergeConfig` switch.

In addition, the `/r` and `/ct` switches for referencing types are for generating data contracts. These switches do not work when using XmlSerializer.

Timeout

The tool has a five minute timeout when retrieving metadata. This timeout only applies to retrieving metadata over the network. It does not apply to any processing of that metadata.

Multi-targeting

The tool does not support multi-targeting. If you want to generate a .NET 4 artifact from *svcutil.exe*, use the *svcutil.exe* from the .NET 4 SDK. To generate a .NET 3.5 artifact, use the executable from the .NET 3.5 SDK.

Accessing WSDL Documents

When you use Svcutil to access a WSDL document that has a reference to a security token service (STS), Svcutil makes a WS-MetadataExchange call to the STS. However, the service can expose its WSDL

documents using either WS-MetadataExchange or HTTP GET. Therefore, if the STS has only exposed the WSDL document using HTTP GET, a client written in WinFX will fail. For clients written in .NET Framework 3.5, Svcutil attempts to use both WS-MetadataExchange and HTTP GET to obtain the STS WSDL.

Using SvcUtil.exe

Common Usages

The following table shows some commonly used options for this tool:

OPTION	DESCRIPTION
/directory:<directory>	Directory to create files in. Default: The current directory. Short form: <code>/d</code>
/help	Displays the command syntax and options for the tool. Short form: <code>/?</code>
/noLogo	Suppress the copyright and banner message.
/svcutilConfig:<configFile>	Specifies a custom configuration file to use instead of the App.config file. This can be used to register system.serviceModel extensions without altering the tool's configuration file.
/target:<output type>	Specifies the output to be generated by the tool. Valid values are code, metadata or xmlSerializer. Short form: <code>/t</code>

Code Generation

Svcutil.exe can generate code for service contracts, clients and data types from metadata documents. These metadata documents can be on a durable storage, or be retrieved online. Online retrieval follows either the WS-Metadata Exchange protocol or the DISCO protocol (for details see the Metadata Download section).

You can use the *SvcUtil.exe* tool to generate service and data contracts based on a predefined WSDL document. Use the /serviceContract switch and specify a URL or file location where the WSDL document can be downloaded or found. This generates the service and data contracts defined in the WSDL document that can then be used to implement a complaint service. For more information, see [How to: Retrieve Metadata and Implement a Compliant Service](#).

For a service with a BasicHttpContextbinding endpoint, *Svcutil.exe* generates a BasicHttpBinding with the `allowCookies` attribute set to `true` instead. The cookies are used for context on the server. If you would like to manage context on the client when the service uses cookies, you can manually modify the configuration to use a context binding.

Caution

Svcutil.exe generates the client based on the WSDL or policy file received from the service. The user principal name (UPN) is generated by concatenating username, "@" and a fully-qualified domain name (FQDN). However, for users who registered on Active Directory, this format is not valid and the UPN

generated by the tool causes a failure in Kerberos authentication with the error message "The logon attempt failed". To resolve this problem, you should manually fix the client file generated by this tool.

```
svcutil.exe [/t:code] <metadataDocumentPath>* | <url>* | <epr>
```

ARGUMENT	DESCRIPTION
<code>epr</code>	The path to an XML file that contains a WS-Addressing EndpointReference for a service endpoint that supports WS-Metadata Exchange. For more information, see the Metadata Download section.
<code>metadataDocumentPath</code>	<p>The path to a metadata document (<i>wsdl</i> or <i>xsd</i>) that contains the contract to import into code (.wsdl, .xsd, .wspolicy, or .ws mex).</p> <p>Svcutil follows imports and includes when you specify a remote URL for metadata. However, if you want to process metadata files on the local file system, you must specify all files in this argument. In this way, you can use Svcutil in a build environment where you cannot have network dependencies. You can use wildcards (*.xsd, *.wsdl) for this argument.</p>
<code>url</code>	The URL to a service endpoint that provides metadata or to a metadata document hosted online. For more information on how these documents are retrieved, see the Metadata Download section.

OPTION	DESCRIPTION
<code>/async</code>	<p>Generates both synchronous and asynchronous method signatures.</p> <p>Default: generate only synchronous method signatures.</p> <p>Short Form: <code>/a</code></p>
<code>/collectionType:<type></code>	<p>Specifies the list collection type for a WCF client.</p> <p>Default: collection type is System.Array.</p> <p>Short Form: <code>/ct</code></p>
<code>/config:<configFile></code>	<p>Specifies the filename for the generated configuration file.</p> <p>Default: output.config</p>
<code>/dataContractOnly</code>	<p>Generates code for data contract types only. Service Contract types are not generated.</p> <p>You should only specify local metadata files for this option.</p> <p>Short Form: <code>/dconly</code></p>

OPTION	DESCRIPTION
/enableDataBinding	<p>Implements the INotifyPropertyChanged interface on all Data Contract types to enable data binding.</p> <p>Short Form: <code>/edb</code></p>
/excludeType:<type>	<p>Specifies a fully-qualified or assembly-qualified type name to be excluded from referenced contract types.</p> <p>When using this switch together with <code>/r</code> from separate DLLs, the full name of the XSD class is referenced.</p> <p>Short Form: <code>/et</code></p>
/importXmlTypes	<p>Configures the Data Contract serializer to import non-Data Contract types as IXmlSerializable types.</p>
/internal	<p>Generates classes that are marked as internal. Default: generate public classes only.</p> <p>Short Form: <code>/i</code></p>
/language:<language>	<p>Specifies the programming language to use for code generation. You should provide either a language name registered in the Machine.config file, or the fully qualified name of a class that inherits from CodeDomProvider.</p> <p>Values: c#, cs, csharp, vb, visualbasic, c++, cpp</p> <p>Default: csharp</p> <p>Short form: <code>/l</code></p>
/mergeConfig	<p>Merges the generated configuration into an existing file, instead of overwriting the existing file.</p>
/messageContract	<p>Generates Message Contract types.</p> <p>Short Form: <code>/mc</code></p>
/namespace:<string,string>	<p>Specifies a mapping from a WSDL or XML Schema targetNamespace to a CLR namespace. Using '*' for the targetNamespace maps all targetNamespaces without an explicit mapping to that CLR namespace.</p> <p>To make sure that the message contract name does not collide with operation name, you should either qualify the type reference with <code>::</code>, or make sure the names are unique.</p> <p>Default: Derived from the target namespace of the schema document for Data Contracts. The default namespace is used for all other generated types.</p> <p>Short Form: <code>/n</code> Note: When generating types to use with XmlSerializer, only a single namespace mapping is supported. All generated types will either be in the default namespace or the namespace specified by '*'.</p>

OPTION	DESCRIPTION
/noConfig	Do not generate configuration files.
/noStdLib	Do not reference standard libraries. Default: Mscorlib.dll and System.servicemodel.dll are referenced.
/out:<file>	Specifies the file name for the generated code. Default: Derived from the WSDL definition name, WSDL service name or target namespace of one of the schemas. Short form: <code>/o</code>
/reference:<file path>	References types in the specified assembly. When generating clients, use this option to specify assemblies that might contain types that represent the metadata being imported. You cannot specify message contracts and XmlSerializer types using this switch. If DateTimeOffset referenced, this type is used instead of generating a new type. If the application is written using .NET Framework 3.5, SvcUtil.exe references DateTimeOffset automatically. Short Form: <code>/r</code>
/serializable	Generates classes marked with the Serializable Attribute. Short Form: <code>/s</code>
/serviceContract	Generate code for service contracts only. Client class and configuration will not be generated Short Form: <code>/sc</code>
/serializer:Auto	Automatically select the serializer. This tries to use the Data Contract serializer and uses the XmlSerializer if that fails. Short Form: <code>/ser</code>
/serializer:DataContractSerializer	Generates data types that use the Data Contract Serializer for serialization and deserialization. Short Form: <code>/ser:DataContractSerializer</code>
/serializer:XmlSerializer	Generates data types that use the XmlSerializer for serialization and deserialization. Short Form: <code>/ser:XmlSerializer</code>

OPTION	DESCRIPTION
/targetClientVersion	<p>Specify which version of .NET Framework the application is targeting. Valid values are <code>Version30</code> and <code>Version35</code>. The default value is <code>Version30</code>.</p> <p>Short Form: <code>/tcv</code></p> <p><code>Version30</code>: Use <code>/tcv:Version30</code> if you are generating code for clients that use WinFX.</p> <p><code>Version35</code>: Use <code>/tcv:Version35</code> if you are generating code for clients that use .NET Framework 3.5. When using <code>/tcv:Version35</code> with the <code>/async</code> switch, both event-based and callback/delegate-based asynchronous methods are generated. In addition, support for LINQ-enabled DataSets and DateTimeOffset is enabled.</p>
/wrapped	<p>Controls whether special-casing is used for document-literal styled documents with wrapped parameters. Use the /wrapped switch with the Service Model Metadata Utility Tool (Svcutil.exe) tool to specify normal casing.</p>

NOTE

When the service binding is one of the system-provided bindings (see [System-Provided Bindings](#)), and the [ProtectionLevel](#) property is set to either `None` or `Sign`, Svcutil generates a configuration file using the `<customBinding>` element instead of the expected system-provided element. For example, if the service uses the `<wsHttpBinding>` element with the `ProtectionLevel` set to `Sign`, the generated configuration has `<customBinding>` in the bindings section instead of `<wsHttpBinding>`. For more information about the protection level, see [Understanding Protection Level](#).

Metadata Export

Svcutil.exe can export metadata for services, contracts and data types in compiled assemblies. To export metadata for a service, you must use the `/serviceName` option to specify the service you would like to export. To export all data contract types within an assembly, you should use the `/dataContractOnly` option. By default, metadata is exported for all service contracts in the input assemblies.

```
svcutil.exe [/t:metadata] [/serviceName:<serviceConfigName>] [/dataContractOnly] <assemblyPath>*
```

ARGUMENT	DESCRIPTION
<code>assemblyPath</code>	Specifies the path to an assembly that contains services, contracts or data contract types to be exported. Standard command line wildcards can be used to provide multiple files as input.
OPTION	DESCRIPTION

OPTION	DESCRIPTION
/serviceName:<serviceConfigName>	Specifies the configuration name of a service to be exported. If this option is used, an executable assembly with an associated configuration file must be passed as input. Svcutil.exe searches all associated configuration files for the service configuration. If the configuration files contain any extension types, the assemblies that contain these types must either be in the GAC or explicitly provided using the <code>/reference</code> option.
/reference:<file path>	<p>Adds the specified assembly to the set of assemblies used for resolving type references. If you are exporting or validating a service that uses 3rd-party extensions (Behaviors, Bindings and BindingElements) registered in configuration, use this option to locate extension assemblies that are not in the GAC.</p> <p>Short Form: <code>/r</code></p>
/dataContractOnly	<p>Operates on data contract types only. Service Contracts are not processed.</p> <p>You should only specify local metadata files for this option.</p> <p>Short Form: <code>/donly</code></p>
/excludeType:<type>	<p>Specifies the fully-qualified or assembly-qualified name of a type to be excluded from export. This option can be used when exporting metadata for a service, or a set of service contracts to exclude types from being exported. This option cannot be used together with the <code>/donly</code> option.</p> <p>When you have a single assembly containing multiple services, and each uses separate classes with the same XSD name, you should specify the service name instead of the XSD class name for this switch.</p> <p>XSD or data contract types are not supported.</p> <p>Short Form: <code>/et</code></p>

Service Validation

Validation can be used to detect errors in service implementations without hosting the service. You must use the `/serviceName` option to indicate the service you want to validate.

```
svcutil.exe /validate /serviceName:<serviceConfigName> <assemblyPath>*
```

ARGUMENT	DESCRIPTION
<code>assemblyPath</code>	Specifies the path to an assembly that contains service types to be validated. The assembly must have an associated configuration file to provide service configuration. Standard command-line wildcards can be used to provide multiple assemblies.

OPTION	DESCRIPTION
/validate	<p>Validates a service implementation specified by the <code>/serviceName</code> option. If this option is used, an executable assembly with an associated configuration file must be passed as input.</p> <p>Short Form: <code>/v</code></p>
/serviceName: <serviceConfigName>	<p>Specifies the configuration name of a service to be validated. Svcutil.exe searches all associated configuration files of all input assemblies for the service configuration. If the configuration files contain any extension types, the assemblies that contains these types must either be in the GAC or explicitly provided using the <code>/reference</code> option.</p>
/reference: <file path>	<p>Adds the specified assembly to the set of assemblies used for resolving type references. If you are exporting or validating a service that uses 3rd-party extensions (Behaviors, Bindings and BindingElements) registered in configuration, use this option to locate extension assemblies that are not in the GAC.</p> <p>Short Form: <code>/r</code></p>
/dataContractOnly	<p>Operates on data contract types only. Service Contracts are not processed.</p> <p>You should only specify local metadata files for this option.</p> <p>Short Form: <code>/dconly</code></p>
/excludeType: <type>	<p>Specifies the fully-qualified or assembly-qualified name of a type to be excluded from validation.</p> <p>Short Form: <code>/et</code></p>

Metadata Download

Svcutil.exe can be used to download metadata from running services, and save the metadata to local files. To download metadata, you must specify the `/t:metadata` option. Otherwise, client code is generated. For HTTP and HTTPS URL schemes, Svcutil.exe attempts to retrieve metadata using WS-Metadata Exchange and DISCO. For all other URL schemes, Svcutil.exe only uses WS-Metadata Exchange.

Svcutil issues the following metadata requests simultaneously to retrieve metadata.

- MEX (WS-Transfer) request to the supplied address
- MEX request to the supplied address with `/mex` appended
- DISCO request (using the DiscoveryClientProtocol from ASMX) to the supplied address.

By default, Svcutil.exe uses the bindings defined in the [MetadataExchangeBindings](#) class to make MEX requests. To configure the binding used for WS-Metadata Exchange, you must define a client endpoint in configuration that uses the IMetadataExchange contract. This can be defined either in the configuration file of Svcutil.exe, or in another configuration file specified using the `/svcutilconfig` option.

```
svcutil.exe /t:metadata <url>* | <epr>
```

ARGUMENT	DESCRIPTION
<code>url</code>	The URL to a service endpoint that provides metadata or to a metadata document hosted online.
<code>epr</code>	The path to an XML file that contains a WS-Addressing EndpointReference for a service endpoint that supports WS-Metadata Exchange.

XmlSerializer Type Generation

Services and client applications that use data types that are serializable using the [XmlSerializer](#) generate and compile serialization code for those data types at runtime, which can result in slow start-up performance.

NOTE

Pre-generated serialization code can only be used in client applications and not in services.

Svcutil.exe can generate the necessary C# serialization code from the compiled assemblies for the application, thus improving start-up performance for these applications. For more information, see [How to: Improve the Startup Time of WCF Client Applications using the XmlSerializer](#).

NOTE

Svcutil.exe only generates code for types used by Service Contracts found in the input assemblies.

```
svcutil.exe /t:xmlserializer <assemblyPath>*
```

ARGUMENT	DESCRIPTION
<code>assemblyPath</code>	Specifies the path to an assembly that contains service contract types. Serialization types are generated for all Xml Serializable types in each contract.

OPTION	DESCRIPTION
<code>/reference:<file path></code>	<p>Adds the specified assembly to the set of assemblies used for resolving type references.</p> <p>Short Form: <code>/r</code></p>
<code>/excludeType:<type></code>	<p>Specifies the fully-qualified or assembly-qualified name of a type to be excluded from export or validation.</p> <p>Short Form: <code>/et</code></p>
<code>/out:<file></code>	<p>Specifies the filename for the generated code. This option is ignored when multiple assemblies are passed as input to the tool.</p> <p>Default: Derived from the assembly name.</p> <p>Short Form: <code>/o</code></p>

OPTION	DESCRIPTION
/UseSerializerForFaults	Specifies that the XmlSerializer should be used for reading and writing faults, instead of the default DataContractSerializer .

Examples

The following command generates client code from a running service or online metadata documents.

```
svcutil http://service/metadataEndpoint
```

The following command generates client code from local metadata documents.

```
svcutil *.wsdl *.xsd /language:C#
```

The following command generates data contract types in Visual Basic from local schema documents.

```
svcutil /donly *.xsd /language:VB
```

The following command downloads metadata documents from running services.

```
svcutil /t:metadata http://service/metadataEndpoint
```

The following command generates metadata documents for service contracts and associated types in an assembly.

```
svcutil myAssembly.dll
```

The following command generates metadata documents for a service, and all associated service contracts and data types in an assembly.

```
svcutil myServiceHost.exe /serviceName:myServiceName
```

The following command generates metadata documents for data types in an assembly.

```
svcutil myServiceHost.exe /donly
```

The following command verifies service hosting.

```
svcutil /validate /serviceName:myServiceName myServiceHost.exe
```

The following command generates serialization types for [XmlSerializer](#) types used by any service contracts in the assembly.

```
svcutil /t:xmlserializer myContractLibrary.exe
```

Maximum Nametable Character Count Quota

When using svcutil to generate metadata for a service, you may get the following message:

Error: Cannot obtain Metadata from `http://localhost:8000/somesservice/mex` The maximum nametable character count quota (16384) has been exceeded while reading XML data. The nametable is a data structure used to store strings encountered during XML processing - long XML documents with non-repeating element names, attribute names and attribute values may trigger this quota. This quota may be increased by changing the MaxNameTableCharCount property on the XmlDictionaryReaderQuotas object used when creating the XML reader.

This error can be caused by a service that returns a large WSDL file when you request its metadata. The problem is that the character quota for the svcutil.exe tool is exceeded. This value is set to help prevent

denial of service (dos) attacks. You can increase this quota by specifying the following config file for svcutil.

The following config file shows how to set the reader quotas for svcutil

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <bindings>
      <customBinding>
        <binding name="MyBinding">
          <textMessageEncoding>
            <readerQuotas maxDepth="2147483647" maxStringContentLength="2147483647"
              maxArrayLength="2147483647" maxBytesPerRead="2147483647"
              maxNameTableCharCount="2147483647" />
          </textMessageEncoding>
          <httpTransport maxReceivedMessageSize="2147483647" maxBufferSize="2147483647" />
        </binding>
      </customBinding>
    </bindings>
    <client>
      <endpoint binding="customBinding" bindingConfiguration="MyBinding"
        contract="IMetadataExchange"
        name="http" />
    </client>
  </system.serviceModel>
</configuration>
```

Create a new file called svcutil.exe.config and copy the XML example code into it. Then place the file in the same directory as svcutil.exe. The next time svcutil.exe is run it will pick up the new settings.

Security Concerns

You should use the appropriate Access Control List (ACL) to protect Svcutil.exe's installation folder, Svcutil.config, and files being pointed to by `/svcutilConfig`. This can prevent malicious extensions from being registered and run.

In addition, to minimize the chance that security be compromised, you should not add untrusted extensions to be part of the system or use untrusted code providers with Svcutil.exe.

Finally, you should not use the tool in the middle-tier of your application, as it may cause denial-of-service to the current process.

See Also

- [DataContractAttribute](#)
- [DataMemberAttribute](#)
- [How to: Create a Client](#)

Find Private Key Tool (FindPrivateKey.exe)

5/4/2018 • 2 minutes to read • [Edit Online](#)

This command-line tool can be used to retrieve a private key from a certificate store. For example, *FindPrivateKey.exe* can be used to find the location and name of the private key file associated with a specific X.509 certificate in the certificate store.

IMPORTANT

The FindPrivateKey tool is shipped as a WCF sample. For more information about where to find the sample and how to build it, see [FindPrivateKey](#).

Syntax

```
FindPrivateKey<storeName> <storeLocation> [{ {-n <subjectName>} | {-t <thumbprint>} } [-f | -d | -a]]
```

Remarks

The following tables describe the arguments and the options that can be used with the Find Private Key tool (FindPrivateKey.exe).

ARGUMENT	DESCRIPTION
<code>storeName</code>	Name of the certificate store.
<code>storeLocation</code>	The location of the certificate store.

OPTION	DESCRIPTION
<code>/n < subjectName ></code>	Specifies the subject name of the certificate.
<code>/t < thumbprint ></code>	Specifies the thumbprint of the certificate. Use Certmgr.exe to retrieve the thumbprint of the certificate.
<code>/f</code>	Outputs the file name only.
<code>/d</code>	Outputs the directory only.
<code>/a</code>	Outputs the absolute file name.

Examples

The following command retrieves the private key for John Doe:

```
FindPrivateKey My CurrentUser -n "CN=John Doe"
```

The following command retrieves the private key for the local machine:

```
FindPrivateKey My LocalMachine -t "03 33 98 63 d0 47 e7 48 71 33 62 64 76 5c 4c 9d 42 1d 6b 52" -a
```

ServiceModel Registration Tool (ServiceModelReg.exe)

5/5/2018 • 2 minutes to read • [Edit Online](#)

This command-line tool provides the ability to manage the registration of WCF and WF components on a single machine. Under normal circumstances you should not need to use this tool as WCF and WF components are configured when installed. But if you are experiencing problems with service activation, you can try to register the components using this tool.

Syntax

```
ServiceModelReg.exe[(-ia|-ua|-r)|((-i|-u) -c:<command>)] [-v|-q] [-nologo] [-?]
```

Remarks

The tool can be found in the following location:

%SystemRoot%\Microsoft.Net\Framework\v3.0\Windows Communication Foundation\

NOTE

When the ServiceModel Registration Tool is run on Windows Vista, the **Windows Features** dialog may not reflect that the **Windows Communication Foundation HTTP Activation** option under **Microsoft .NET Framework 3.0** is turned on. The **Windows Features** dialog can be accessed by clicking **Start**, then click **Run** and then typing **OptionalFeatures**.

The following tables describe the options that can be used with ServiceModelReg.exe.

OPTION	DESCRIPTION
-ia	Installs all WCF and WF components.
-ua	Uninstalls all WCF and WF components.
-r	Repairs all WCF and WF components.
-i	Installs WCF and WF components specified with -c.
-u	Uninstalls WCF and WF components specified with -c.

OPTION	DESCRIPTION
<code>-c</code>	Installs or uninstalls a component: <ul style="list-style-type: none"> - httpnamespace – HTTP Namespace Reservation - tcpportsharing – TCP port sharing service - tcpactivation – TCP activation service (unsupported on .NET 4 Client Profile) - namedpipeactivation – Named pipe activation service (unsupported on .NET 4 Client Profile) - msmqactivation – MSMQ activation service (unsupported on .NET 4 Client Profile) - etw – ETW event tracing manifests (Windows Vista or later)
<code>-q</code>	Quiet mode (only display error logging)
<code>-v</code>	Verbose mode.
<code>-nologo</code>	Suppresses the copyright and banner message.
<code>-?</code>	Displays help text

Fixing the FileLoadException Error

If you installed previous versions of WCF on your machine, you may get a `FileLoadFoundException` error when you run the ServiceModelReg tool to register a new installation. This can happen even if you have manually removed files from the previous install, but left the machine.config settings intact.

The error message is similar to the following.

```
Error: System.IO.FileLoadException: Could not load file or assembly 'System.ServiceModel, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089' or one of its dependencies. The located assembly's manifest definition does not match the assembly reference. (Exception from HRESULT: 0x80131040)
File name: 'System.ServiceModel, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089'
```

You should note from the error message that the System.ServiceModel Version 2.0.0.0 assembly was installed by an early Customer Technology Preview (CTP) release. The current version of the System.ServiceModel assembly released is 3.0.0.0 instead. Therefore, this issue is encountered when you want to install the official WCF release on a machine where an early CTP release of WCF was installed, but not completely uninstalled.

ServiceModelReg.exe cannot clean up prior version entries, nor can it register the new version's entries. The only workaround is to manually edit machine.config. You can locate this file at the following location.

```
%windir%\Microsoft.NET\Framework\v2.0.50727\config\machine.config
```

If you are running WCF on a 64-bit machine, you should also edit the same file at this location.

```
%windir%\Microsoft.NET\Framework64\v2.0.50727\config\machine.config
```

Locate any XML nodes in this file that refer to "System.ServiceModel, Version=2.0.0.0", delete them and any child nodes. Save the file and re-run ServiceModelReg.exe resolves this problem.

Examples

The following examples show how to use the most common options of the ServiceModelReg.exe tool.

```
ServiceModelReg.exe -ia
    Installs all components
ServiceModelReg.exe -i -c:httpnamespace -c:etw
    Installs HTTP namespace reservation and ETW manifests
ServiceModelReg.exe -u -c:etw
    Uninstalls ETW manifests
ServiceModelReg.exe -r
    Repairs an extended install
```

Service Trace Viewer Tool (SvcTraceViewer.exe)

8/31/2018 • 28 minutes to read • [Edit Online](#)

Windows Communication Foundation (WCF) Service Trace Viewer Tool helps you analyze diagnostic traces that are generated by WCF. Service Trace Viewer provides a way to easily merge, view, and filter trace messages in the log so that you can diagnose, repair, and verify WCF service issues.

Configuring Tracing

Diagnostic traces provide you with information that shows what is happening throughout your application's operation. As the name implies, you can follow operations from their source to destination and through intermediate points as well.

You can configure tracing using the application's configuration file—either `Web.config` for Web-hosted applications, or `Appname.config` for self-hosted applications. The following is an example:

```
<system.diagnostics>
  <trace autoflush="true" />
  <sources>
    <source name="System.ServiceModel"
      switchValue="Information, ActivityTracing"
      propagateActivity="true">
      <listeners>
        <add name="sdt"
          type="System.Diagnostics.XmlWriterTraceListener"
          initializeData= "SdrConfigExample.e2e" />
      </listeners>
    </source>
  </sources>
</system.diagnostics>
```

In this example, the name and type of the trace listener is specified. The Listener is named `sdt` and the standard .NET Framework trace listener (`System.Diagnostics.XmlWriterTraceListener`) is added as the type. The `initializeData` attribute is used to set the name of the log file for that Listener to be `SdrConfigExample.e2e`. For the log file, you can substitute a fully-qualified path for a simple file name.

The example creates a file in the root directory called `SdrConfigExample.e2e`. When you use the Trace Viewer to open the file as described in the "Opening and Viewing WCF Trace Files" section, you can see all the messages that have been sent.

The tracing level is controlled by the `switchValue` setting. The available tracing levels are described in the following table.

TRACE LEVEL	DESCRIPTION
Critical	<ul style="list-style-type: none">- Logs Fail-Fast and Event Log entries, and trace correlation information. The following are some examples of when you might use the Critical level:- Your AppDomain went down because of an unhandled exception.- Your application fails to start.- The message that caused the failure originated from the process MyApp.exe.

TRACE LEVEL	DESCRIPTION
Error	<ul style="list-style-type: none"> - Logs all exceptions. You can use the Error level in the following situations: - Your code crashed because of an Invalid Cast Exception. - A "failed to create endpoint" exception is causing your application to fail on startup.
Warning	<ul style="list-style-type: none"> - A condition exists that may subsequently result in an error or critical failure. You can use this level in the following situations: - The application is receiving more requests than its throttling settings allows. - The receiving queue is at 98 percent of its configured capacity.
Information	<ul style="list-style-type: none"> - Messages helpful for monitoring and diagnosing system status, measuring performance, or profiling are generated. You can utilize such information for capacity planning and performance management. You can use this level in the following situations: - A failure occurred after the message reached the AppDomain and was deserialized. - A failure occurred while the HTTP binding was being created.
Verbose	<ul style="list-style-type: none"> - Debug-level tracing for both user code and servicing. Set this level when: - You are not sure which method in your code was called when the failure occurred. - You have an incorrect endpoint configured and the service failed to start because the entry in the reservation store is locked.
ActivityTracing	<p>Flow events between processing activities and components.</p> <p>This level allows administrators and developers to correlate applications in the same application domain.</p> <ul style="list-style-type: none"> - Traces for activity boundaries: start/stop. - Traces for transfers.

You can use `add` to specify the name and type of the trace listener you want to use. In the example configuration, the Listener is named `sdt` and the standard .NET Framework trace listener (`System.Diagnostics.XmlWriterTraceListener`) is added as the type. Use `initializeData` to set the name of the log file for that Listener. In addition, you can substitute a fully-qualified path for a simple file name.

Using the Service Trace Viewer Tool

Opening and Viewing WCF Trace Files

The Service Trace Viewer supports three file types:

- WCF Tracing File (.svcLog)
- Event Tracing File (.etl)
- Crimson Tracing File

Service Trace Viewer enables you to open any supported trace file, add and integrate additional trace files, or open and merge a group of trace files simultaneously.

To open a trace file

1. Start Service Trace Viewer by using a command window to navigate to your WCF installation location (C:\Program Files\Microsoft SDKs\Windows\v6.0\Bin), and then type `SvcTraceViewer.exe`.

NOTE

The Service Trace Viewer tool can associate with two file types: .svclog and .stvproj. You can use two parameters in command line to register and unregister the file extensions.

/register: register the association of file extensions ".svclog" and ".stvproj" with SvcTraceViewer.exe

/unregister: unregister the association of file extensions ".svclog" and ".stvproj" with SvcTraceViewer.exe

1. When Service Trace Viewer starts, click **File** and then point to **Open**. Navigate to the location where your trace files are stored.
2. Double-click the trace file that you want to open.

NOTE

Press SHIFT while clicking multiple trace files to select and open them simultaneously. Service Trace Viewer merges the content of all files and presents one view. For example, you can open trace files of both client and service. This is useful when you have enabled message logging and activity propagation in configuration. In this way, you can examine message exchange between client and service. You can also drag multiple files into the viewer, or use the **Project** tab. See the Managing Project section for more details.

3. To add additional trace files to the collection that is open, click **File** and then point to **Add**. In the window that opens, navigate to the location of the trace files and double-click the file you want to add.

Caution

It is not recommended that you load a trace log file bigger than 200MB. If you attempt to load a file larger than this limit, the loading process may take a long time, depending on your computer resource. The Service Trace Viewer tool may not be responsive for a long time, or it may exhaust your machine's memory. It is recommended that you configure partial loading to avoid this. For more information on how to do this, see "Loading Large Trace Files" section.

Event Tracing and Crimson Tracing

The viewer's native format is the activity tracing format that WCF emits. Traces emitted in a different format must be converted before the viewer displays them. Currently, in addition to the activity tracing format, the viewer supports event tracing and crimson tracing.

When you open a file that does not contain activity traces, the viewer attempts to convert the file. You must specify the name and location of the file that will contain the converted trace data. Once the data has been converted, the viewer displays the content of the new file.

NOTE

Conversion requires disk space to store the converted trace data. Make sure you have enough disk space available to store the data before you start a conversion. Otherwise, the conversion fails.

Managing Projects

The viewer supports projects to facilitate viewing multiple trace files. For example, if you have a client trace file and a service trace file, you can add them to a project. Then, every time you open the project, all the trace files in the project are loaded simultaneously.

There are two ways to manage projects:

- In the **File** menu, you can open, save and close projects.
- In the **Project** tab, you can add files to a project.

Viewing WCF Traces

WCF emits traces using the activity tracing format. In the activity tracing model, individual traces are grouped in activities according to their purpose. Logical control flow is transferred between activities. For example, during the lifetime of an application, many "message send activities" appear and disappear. For more information on viewing traces and activities, and the user interface of the Service Trace Viewer too, see [Using Service Trace Viewer for Viewing Correlated Traces and Troubleshooting](#).

Switching to Different Views

The Service Trace Viewer provides the following different views. They are displayed as tabs on the left pane of the Viewer, and can also be accessed from the **View** menu.

- Activity View
- Project View
- Message View
- Graph View

Activity view

Once the trace files are opened, you can see the traces grouped into activities and displayed in the **Activity** view in the left-hand pane.

The **Activity** view displays activity names, number of traces in the activity, duration time, start time and end time.

By clicking any of the listed activities, the traces in this activity are displayed in the trace pane on the right. You can then select a trace to view its details.

You can select multiple activities by pressing the **Ctrl** or **Shift** key and clicking the desired activities. The trace pane displays all the traces of the selected activities.

You can double-click an activity to display it in **Graph** View. The alternative way is to select an activity and switch to **Graph** View.

NOTE

The activity "000000000000" is a special activity that cannot be displayed in the Graph View. Because all other activities are linked to it, displaying this activity has a severe performance impact.

You can click the column title to sort the activity list. Activities that contain warning traces have a yellow background and those that contain error traces have a red one.

There are different types of activities and each type corresponds to an icon on the left side of each activity. You can refer to the Understanding Trace Icons section for their meaning.

Project View

This view enables you to manage trace files in the current project. See the Managing Project section for more details.

Graph View

One of the most powerful features of Service Trace Viewer is the **Graph** view, which displays the trace data for a given activity in chart form. The chart form enables you to see the stepwise execution of events and the interrelationships between multiple activities as data moves between them.

To switch to **Graph** view, select an activity in the **Activity** view and click the **Activity** tab, or a message log trace in the **Message** View. If multiple trace files are loaded and the activity involves traces from more than one file, all of

the relevant traces appear in the graph view. Double-clicking on the activities and message log traces also leads you to the **Graph** view.

In **Graph** view, each vertical column represents an activity, and each block in the column represents a trace. The activities are grouped by process (or thread). The small arrows between activities represent transfers. The big arrows between processes represent message exchange. The activity in selection is always in yellow.

Selecting Traces in the Graph

1. Click a block in the graph.
2. Use the up and down keys to select its neighboring traces.
3. Observe the trace information in the Trace Pane and Detail Pane.

Expanding or Collapsing Activity Transfers

You can expand activity transfers when the activity in selection transfers out to another activity. It enables you to follow the transfers.

To expand or collapse activity transfers,

1. Locate the transfer trace with a "+" sign on the left of the transfer icon.
2. Click the "+", or press **Ctrl** and "+" using the keyboard.
3. The next activity appears in the graph.
4. A "-" appears on the left of the transfer icon. Click the "-" sign or press Ctrl and "-", the activity transfer collapses.

NOTE

When an activity has multiple transfers into it and you expand one of the transfers, activities that lead up to the new activity from the root activity are displayed. These new activities appear in collapsed form. If you want to see the details of these activities, expand them vertically by clicking the expand icon in the header of the graph.

Expanding or Collapsing Activities Vertically

The viewer hides unnecessary detail in the activity graph by collapsing activities. In a collapsed activity, individual traces are not displayed. Only transfers trace appear. If you want to view all traces in an activity, expand the activity vertically by clicking the expand symbol of the activity in the header of the graph.

To expand or collapse activities vertically,

1. Click the "+" icon in the activity header to expand the activity vertically.
2. Notice that all traces are displayed in the graph.
3. Click the "-" icon in the activity header to collapse the activity vertically.
4. Notice that only important transfers, message logs, warning and exception traces are shown in the activity.

Options

You can select two options from the **Option** menu in Graph view.

- Show Activity Boundary Traces, which when unchecked ignore the activity boundary traces in the graph.
- Show Non-message Verbose Traces, which when unchecked ignore verbose level traces, except for message traces. In most cases, verbose level traces are less important for analysis. This option is helpful when you do not want to analyze verbose level traces and only want to focus on more important traces.

Layout Mode

The viewer has two Layout Modes: **Process** and **Thread**. This setting defines the largest unit of organization. The default Layout Mode is **Process**, which means that activities are grouped by processes in the graph.

Execution List

You can select which process or thread to be displayed in the graph from this drop-down list. For example, if you have the trace files of two clients (A and B) and one service opened, and you only want to display the service and client A in the graph, you can deselect client B from the list.

Viewing Trace Details

To view a trace detail, select a trace in the Trace pane. The details are displayed in the Detail pane.

Trace Pane

The upper right pane in the Service Trace Viewer is the Trace Pane. It lists all the traces in the selected activity with extra information, for example, trace level, thread ID, and process name.

You can copy the raw XML of the trace to the clipboard by right-clicking a trace and selecting **Copy Trace to Clipboard**.

Detail Pane

The bottom left pane in the Service Trace Viewer is the Detail Pane. It provides three tabs to view trace details.

The **Formatted** view displays the information in a more organized way. It lists all known XML elements in tables and trees, making it easier to read and understand the information.

The **XML** view displays XML corresponding to the selected trace. It supports highlighting and syntax color. When you use **Find** to search strings, it highlights the search results.

The **Message** view displays the message part of the XML in message log traces. It is invisible when you select a non-message trace.

Filtering WCF Traces

To make the analysis of trace easier, you can filter them in the following ways:

- The filter toolbar provides access to pre-defined and custom filters. It can be enabled through the **View** menu.
- The pre-defined filter of the viewer can be used to selectively filter parts of the WCF traces. By default, it is set to allow all infrastructure traces to pass through. The settings of this filter are defined in the **Filter Options** sub-menu under **View** menu.
- Custom XPath filters give users full control over filtering. They can be defined in the **Custom Filter** under **View** menu.

Only traces that passes through all filters is displayed.

Using the Filter Toolbar

The filter toolbar appears across the top of the tool. If it is not present, you can activate it in the **View** menu. The bar has three components:

- Look for: **Look for** defines the subject to look for in the filter operation. For example, if you want to find all traces that were emitted in the context of process X, set this field to X and the **Search In** field to 'Process Name'. This field changes to a DateTime selector control when a time-based filter is selected.
- Search in: This field defines the type of filter to apply.
- Level: The level setting defines the minimum trace level allowed by the filter. For example, if the level is set to Error and Up, only traces at the Error and critical level are displayed. This filter combines with the criteria specified by Look For and Search In.

The **Filter Now** button starts the filter operation. Some filters, especially when they are applied to a large data set, take a long time to complete. You can cancel the filter operation by pressing the **Stop** button that appears in the status bar under the **Operations** menu.

The **Clear** button resets pre-defined and custom filters to allow all traces to pass through.

Filter Options

The viewer can automatically remove WCF traces from the view. It can selectively remove traces emitted by specific areas of WCF, for example, removing transaction related traces from the view.

The settings of this filter are defined in the **Filter Options** sub-menu under **View** menu.

Custom Filters

If you are familiar with the XML Path Language (XPath), you can use it to construct custom filters to search the trace data for any XML element of interest. The filters are accessible through the filter toolbar.

Custom filters can include parameters. You can also import pre-existing custom filters.

Creating a custom filter

Filters can be created in two ways:

Creating a Custom Filter using the Template Wizard

You can click an existing trace and create a filter based on the structure of the trace. This example creates a custom filter based on thread ID.

1. In the trace pane in the top right area of the viewer, select a trace that includes the element you want to filter for.
2. Click the **Create Custom Filter** button located at the top of the trace pane.
3. In the dialog box that appears, enter a name for your filter. In this example, enter `Thread ID`. You can also provide a description of your filter.
4. The tree view on the left displays the structure of the trace record you selected in step 1. Browse to the element you want to create a condition for. In this example, browse to the ThreadID to be located in the XPath: `/E2ETraceEvent/System/Execution/@ThreadID` node. Double-click the ThreadID attribute in the tree view. This creates an expression for the attribute on the right of the dialog.
5. Change the parameter field for the ThreadID condition from None to `{0}`. This step enables the ThreadID value to be configured when the filter is applied. (See the How to Apply a Filter section) You can define up to four parameters. Conditions are combined using the OR operator.
6. Click **Ok** to create the filter.

NOTE

Once a filter has been created using the template wizard, it can only be edited manually. It is not possible to activate the wizard for a filter that has been created previously. In addition, the conditions of an XPath filter created in the template wizard are combined using the OR operator. If you require an AND operation, you can edit the filter expression after it has been created.

Creating a Custom Filter Manually

The Custom Filters menu allows you to enter XPath filters manually.

1. In the View menu, click the **Custom Filters** menu item.
2. In the dialog that appears, click **New**.
3. At the minimum, specify a Filter Name and XPath expression.
4. Click **OK**.

Applying a Custom Filter

Once a custom filter has been created, it is accessible through the filter toolbar. Select the filter you want to apply in the **Search In** field of the filter toolbar. For the previous example, select 'Thread ID'.

1. Specify the value you are looking for in the **Find What** field. In our example, enter the ID of the thread you

want to search for.

2. Click **Filter Now**, and observe the result of the operation.

If your filter uses multiple parameters, enter them using ';' as a separator in the **Find What** field. For example, the following string defines 3 parameters: '1;findValue;text'. The viewer applies '1' to the {0} parameter of the filter. 'findValue' and 'text' are applied to {1} and {2} respectively.

Sharing custom Filters

Custom filters can be shared between different sessions and different users. You can export the filters to a definition file and import this file at another location.

To import a custom filter:

1. In the **View** menu, click **Custom Filters**.
2. In the dialog box that opens, click the **Import** button.
3. Navigate to the custom filter file (.stvcf), click the file, and click the **Open** button.

To export a custom filter:

1. In the View menu, click **Custom Filters**.
2. In the dialog box that opens, select the filter you want to export.
3. Click the **Export** button.
4. Specify the name and location of the custom filter definition file (.stvcf), and click the **Save** button.

NOTE

These custom filters can only be imported and exported from Service Trace Viewer. They cannot be read by other tools.

Finding Data

The viewer provides the following ways to find data:

- The Find toolbar provides a quick access to the most common find options.
- The Find dialog provides more find options. It is accessible through the **Edit** menu, or by the short key Ctrl + F.

The find toolbar appears at the top of the viewer. If it is not present, you can activate it in the **View** menu. The bar has two components:

- Find What: Allows you to enter search keyword.
- Look In: Allows you to enter the search scope. You can select whether to search in all activities or in the current activity only.

The find dialog provides two additional options:

- Find target:
 - The "Raw log data" option searches the keyword in all raw data.
 - The "XML Text" and "XML Attribute" options only search in XML elements.
 - The "Logged Message" option searches the keyword only in messages.
- Ignore root activity: The search ignores the traces in the "000000000000" activity. This improves performance in large trace files when the root activity has thousands of traces, most of which are transfers.

Navigating Traces

Because traces are recorded step by step during application runtime, navigating traces can help you to debug your application. The Service Trace Viewer provides various ways to navigate in traces.

Step Forward or Backward

If you consider each trace as a line of code in the program, stepping forward is very similar to "Step over" in the Visual Studio Integrated Development Environment (IDE). The difference is that you can also step backward in the traces. Stepping forward means moving to the next trace in the activity.

- Step Forward: Use the **Activity** menu, or press "F10". You can also use arrow key "down" in the trace pane.
- Step Backward: Use the **Activity** menu, or press "F9". You can also use arrow key "up" in the trace pane.

NOTE

This can take you to an activity occurring in a different process or even on a different computer, because WCF messages can carry activity IDs that span machines.

Follow Transfer

Transfer traces are special traces in the trace file. An activity may transfer to another activity by a transfer trace. For example, "Activity A" may transfer to "Activity B". In such case, there is a transfer trace in the "Activity A" with the name "To: Activity" and the transfer icon. This transfer trace is a link between the two traces. In "Activity B", there might also be a transfer trace at the end of the activity to transfer back to "Activity A". This is similar to function calls in programs: A calls B, then B returns.

"Follow transfer" is similar to "Step into" in a debugger. It follows the transfer from A to B. It does not have any effect on other traces.

There are two ways to follow a transfer: by mouse or by keyboard:

- By Mouse: Double-click the transfer trace in the trace pane.
- By Keyboard: Select a transfer trace, and use "Follow Transfer" in the **Activity** menu, or press "F11"

NOTE

In many cases, when Activity A transfers to Activity B, Activity A waits until Activity B transfers back to Activity A. This means that Activity A has no trace logged during the period when Activity B is actively tracing. However, it is also possible that Activity A does not wait, and continues to log traces. It is also possible that Activity B does not transfer back to Activity A. Therefore, activity transfers are still different from function calls in this sense. You can understand activity transfers better in Graph view.

Jump to Next or Previous Transfer

When you are analyzing the current activity, or selected activities when multiple activities are selected, you may want to quickly find the activities it transfers to. "Jump to next transfer" allows you to locate the next transfer trace in the activity. Once you find the transfer trace, you can use "Follow transfer" to step into the next activity.

- Jump to Next Transfer: Use the **Activity** menu, or press "Ctrl + F10".
- Jump to Previous Transfer: Use the **Activity** menu, or press "Ctrl + F9".

Navigate in Graph View

Although navigating in the activity pane and trace pane is similar to debugging, using **Graph** view provides a much better experience in navigation. See "Graph View" section for more information.

Loading Large Trace Files

Trace files can be very large. For example, if you turn on tracing on the "Verbose" level, the resulting trace file for

running a few minutes can easily be hundreds of megabytes or even larger, depending on network speed and communication pattern.

When you open a very large trace file in the Service Trace Viewer, system performance can be negatively impacted. The loading speed and the response time after loading can be slow. Actual speed differs from time to time, depending on your hardware configuration. In most PCs, loading a trace file larger than 200M has a severe performance impact. For traces files larger than 1G, the tool may use up all available memory, or stop responding for a very long time.

In order to avoid the slow loading and response time in analyzing large trace files, the Service Trace Viewer provides a feature called "Partial Loading", which only loads a small part of the trace at a time. For example, you may have a trace file over 1GB, running for several days on the server. When some errors have occurred and you want to analyze the trace, it is not necessary to open the entire trace file. Instead, you can load the traces within a certain period of time when the error might have occurred. Because the scope is smaller, the Service Trace Viewer tool can load the file faster and you can identify the errors using a smaller set of data.

Enabling Partial Loading

You do not need to manually enable partial loading. If the total size of the trace file(s) you attempt to load exceeds 40MB, Service Trace Viewer automatically displays a Partial Loading dialog for you to select the part that you want to load.

NOTE

Because traces may not be distributed evenly in the time span, the length of the time period you specify in the Partial Loading toolbar may not be proportional to the loading size shown. The actual loading size can be smaller than the Estimated Size in the partial loading dialog.

Adjusting Partial Loading

After you have partially loaded the trace file, you may want to change the data set being loaded. You can do so by adjusting the Partial Loading toolbar at the top of the viewer.

- 1. Move the toolbar by mouse, or input the Begin and End time.
- 2. Click the **Adjust** button.



Understanding Trace Icons








The following is a list of icons that the Service Trace Viewer tool uses in the **Activity** view, **Graph** view and **Trace** pane to represent different items.

NOTE

Some traces that are not categorized (for example, "a message is closed") have no icon.




Activity Tracing Traces

ICON	DESCRIPTION
	Warning trace: A trace that is emitted at the warning level
	Error trace: A trace that is emitted at the error level.



ICON	DESCRIPTION
	<p>Activity Start trace: A trace that marks the beginning of an activity. It contains the name of the activity. As the application designer or developer, you should define one activity Start trace per activity id per process or thread.</p> <p>If the activity id is propagated across trace sources for trace correlation, you can then see multiple Starts for the same activity id (one per trace source). The Start trace is emitted if ActivityTracing is enabled for the trace source.</p>
	<p>Activity Stop trace: A trace that marks the end of an activity. . It contains the name of the activity. As the application designer or developer, you should define one activity Stop trace per activity id per trace source. No traces from a given trace source appear after the activity Stop emitted by that trace source, except if the trace time granularity is not sufficiently small. When that happens, two traces with the same time, including a Stop, may be interleaved when displayed. If the activity id is propagated across trace sources for trace correlation, you can see multiple Stops for the same activity id (one per trace source). The Stop trace is emitted if ActivityTracing is enabled for the trace source.</p>
	<p>Activity Suspend trace: A trace that marks the time an activity is paused. No traces are emitted in a suspended activity until the activity resumes. A suspended activity denotes that no processing is happening in that activity in the scope of the trace source. Suspend/Resume traces are useful for profiling. The Suspend trace is emitted if ActivityTracing is enabled for the trace source.</p>
	<p>Activity resume trace: A trace that marks the time an activity is resumed after it had been suspended. Traces may be emitted again in that activity. Suspend/Resume traces are useful for profiling. The Resume trace is emitted if ActivityTracing is enabled for the trace source.</p>
	<p>Transfer: A trace that is emitted when logical control flow is transferred from one activity to another. The activity the transfer originates from may continue to perform work in parallel to the activity the transfer goes to. The Transfer trace is emitted if ActivityTracing is enabled for the trace source.</p>
	<p>Transfer From: A trace that defines a transfer from another activity to the current activity.</p>
	<p>Transfer To: A trace that defines a transfer of logical control flow from the current activity to another activity.</p>

WCF Traces





ICON	DESCRIPTION
------	-------------



ICON	DESCRIPTION
	Message Log trace: A trace that is emitted when a WCF message is logged by the message logging feature, when the <code>System.ServiceModel.MessageLogging</code> trace source is enabled. Clicking on this trace displays the message. There are four configurable logging points for a message: <code>ServiceLevelSendRequest</code> , <code>TransportSend</code> , <code>TransportReceive</code> , and <code>ServiceLevelReceiveRequest</code> , which can also be specified by the <code>messageSource</code> attribute in the message log trace.
	Message Received trace: A trace that is emitted when a WCF message is received, if the <code>System.ServiceModel</code> trace source is enabled at the Information or Verbose level. This trace is essential for viewing the message correlation arrow in the Activity Graph view.
	Message Sent trace: A trace that is emitted when a WCF message is sent if the <code>System.ServiceModel</code> trace source is enabled at the Information or Verbose level. This trace is essential for viewing the message correlation arrow in the Activity Graph view.

Activities

ICON	DESCRIPTION
	Activity: Indicates that the current activity is a generic activity.
	Root activity: Indicates the root activity of a process.

WCF Activities

ICON	DESCRIPTION
	Environment activity: An activity that creates, opens, or closes a WCF host or client. Errors that have happened during these phases will appear in this activity.
	Listen activity: An activity that logs traces related to a listener. Inside this activity, we can view listener information and connection requests.
	Receive Bytes activity: An activity that groups all traces related to receiving incoming bytes on a connection between two endpoints. This activity is essential in correlating with transport activities that propagate their activity id such as <code>http.sys</code> . Connection errors such as aborts will appear in this activity.
	Process Message activity: An activity that groups traces related to creating a WCF message. Errors due to a bad envelope or a malformed message will appear in that activity. Inside this activity, we can inspect message headers to see if an activity id was propagated from the caller. If this is true, when we transfer to Process Action activity (the next icon), we can also assign to that activity the propagated activity id for correlation between the caller and callee's traces.

ICON	DESCRIPTION
	Process Action activity: An activity that groups all traces related to a WCF request across two endpoints. If <code>propagateActivity</code> is set to <code>true</code> on both endpoints in configuration, all traces from both endpoints are merged into one activity for direct correlation. Such activity will contain errors due to transport or security processing, extending to the user code boundary and back (if a response exists).
	Execute User Code activity: An activity that groups user code traces for processing a request.

Troubleshooting

If you do not have permission to write to the registry, you get the following error message "The Microsoft Service Trace Viewer was not registered to the system" when you use the "`svctraceviewer /register`" command to register the tool. If this occurs, you should log in using an account that has write access to the registry.

In addition, the Service Trace Viewer tool writes some settings (for example, custom filters and filter options) to the `SvcTraceViewer.exe.settings` file in its assembly folder. If you do not have read permission for the file, you can still launch the tool, but you cannot load the settings.

If you get the error message "An unknown error occurred while processing one or more traces" when opening the .etl file, it means that the format of the .etl file is invalid.

If you open a trace log created using an Arabic operating system, you may notice that the time filter does not work. For example, year 2005 corresponds to year 1427 in Arabic calendar. However, the time range supported by the Service Trace Viewer tool filter does not support a date earlier than 1752. This can imply that you are not able to select a correct date in the filter. To resolve this problem, you can create a custom filter (**View/Custom Filters**) using an XPath expression to include a specific time range.

See Also

[Using Service Trace Viewer for Viewing Correlated Traces and Troubleshooting](#)

[Configuring Tracing](#)

[Activity Tracing and Propagation for End-To-End Trace Correlation](#)

Configuration Editor Tool (SvcConfigEditor.exe)

5/5/2018 • 17 minutes to read • [Edit Online](#)

The Windows Communication Foundation (WCF) Service Configuration Editor (SvcConfigEditor.exe) allows administrators and developers to create and modify configuration settings for WCF services using a graphical user interface. With this tool, you can manage settings for WCF bindings, behaviors, services, and diagnostics without having to directly edit XML configuration files.

Service Configuration Editor can be found in the C:\Program Files\Microsoft SDKs\Windows\v6.0\Bin folder.

The WCF Configuration Editor

Service Configuration Editor comes with a wizard that guides you through all the steps in configuring a WCF service or client. You are strongly advised to use the wizard instead of the editor directly.

If you already have some configuration files that comply with the standard System.Configuration schema, you can manage specific settings for bindings, behavior, services, and diagnostics with the user interface. The Service Configuration Editor enables you to manage the settings for existing WCF configuration files as well as executable files, COM+ services, and Web-hosted services. When opening a Web-hosted service with Service Configuration Editor, both the service's own configuration and inherited configurations sections of upper level nodes are shown.

Because WCF configuration settings are located in the `<system.serviceModel>` section of the configuration file, the editor operates exclusively on the content of this element and does not access other elements in the same file. You can navigate to existing configuration files directly or you can select an assembly that contains a service, virtual directory, or COM+ service. The editor loads the configuration file for that particular service and allows the user to either add new elements or edit existing elements nested in the `<system.serviceModel>` section of the configuration file.

The editor supports IntelliSense and enforces schema compliance. The resulting output is guaranteed to comply with the schema of the configuration file and to have syntactically correct data values. However, the editor does not guarantee that the configuration file is semantically valid. In other words, the editor does not guarantee that the configuration file can work with the service it configures.

Caution

The editor cannot purge a configuration element from the configuration file once you have modified the element. For example, if you use the editor to set the endpoint name to a non-empty string and save it, the configuration file has the following content, as shown in the following example.

```
<endpoint binding="basicHttpBinding" name="somename" />
```

If you attempt to remove the name by setting it to an empty string and save the file, the configuration file still contains the `name` attribute, as shown in the following example.

```
<endpoint binding="basicHttpBinding" name="" />
```

To purge the attribute, you must manually edit the element using another text editor.

You should be especially careful with this issue when you use the `issueToken` element of the `clientCredential` Endpoint behavior. Specifically, the `address` attribute of its `localIssuer` sub-element must not be an empty string. If you have modified the `address` attribute using the Configuration Editor and want to remove it completely, you should do so using a tool other than the Editor. Otherwise, the attribute contains an empty string and your application throws an exception.

Using the Configuration Editor

The Service Configuration Editor can be found at the following Windows SDK installation location:

C:\Program Files\Microsoft SDKs\Windows\v6.0\Bin\SvcConfigEditor.exe

After you launch the Service Configuration Editor, you can use the **File/Open** menu to browse for the service or assembly you want to manage. You can open configuration files directly, browse for WCF /COM+ services, and open configuration files for Web-hosted services.

The Service Configuration Editor's user interface is divided into the following areas:

- Tree View Pane, which displays configuration elements in a tree structure on the left. You can perform operations in the tree by right-clicking the nodes.
- Task Pane, which displays common tasks for current elements on the lower-left of the window
- Detail Pane, which displays detailed settings of the configuration node selected in the Tree View on the right.

Opening a Configuration File

1. Start Service Configuration Editor by using a command window to navigate to your WCF installation location, and then type `SvcConfigEditor.exe`.
2. From the **File** menu, select **Open** and click the type of file you want to manage.
3. In the **Open** dialog box, navigate to the specific file you want to manage and double-click it.

The viewer automatically follows the configuration merge path and creates a view of the merged configuration. For example, the actual configuration of a non-hosted service is a combination of Machine.config and App.config. Any changes are applied to the active file in the SvcConfigEditor. If you want to edit a specific file in the configuration merge path, you should open it directly.

NOTE

Configuration Editor reloads the currently opened configuration file when the latter has been modified outside the Editor. When this happens, all the changes that are not durably saved inside the Editor are lost. If reloading happens consistently, the most likely cause is a service that constantly accesses the configuration file, for example, an antivirus software running in the background. To resolve this, ensure that Configuration Editor is the only process that can access the file when it is opened.

Services

The **Services** node displays all of the services currently assigned in the configuration file. Each sub-node in the tree corresponds to a sub-element of the `<services>` element in the configuration file.

When you click the **Services** node, you can view or perform tasks on the service Summary Page in the **Detail** Pane.

Creating a new Service Configuration

You can create a new service configuration in the following ways:

- Using a Wizard: Click the link **Create a New Service...** on the Task Pane or Summary Page to launch the wizard. You can also do so in the **File** menu -> **Add New Item**.
- Create manually: You can right-click the **Services** node and choose **New Service**.

Creating a new Service Endpoint Configuration

You can create a new service endpoint configuration in the following ways:

- Create using a Wizard: click the link **Create a New Service Endpoint...** on the Task Pane or Summary Page to launch the wizard. You can also do so in the **File** menu -> **Add New Item**.
- Create manually: Once you created a Service, you can right-click the **Endpoints** node and choose "**New Service Endpoint**".

Editing a Service Configuration

1. Click a **Service** node.
2. Edit the settings in the property grids.

Editing a Service Endpoint Configuration

1. Click a **Service Endpoint** node.
2. Edit the settings in the property grids.

Adding a Base Address

1. Click the **Host** node.
2. Click the **New...** button in the **Base Addresses** section.
3. Type in the base address URI in the dialog box.
4. Click **OK**.

NOTE

You cannot edit the value of `<baseAddressPrefixFilters>` inside this tool. To add or modify this element, you should use a text editor or Visual Studio.

Client

The **Client** node displays all of the client endpoints in the configuration file. Every sub-node in the tree corresponds to a sub-element of the `<client>` element in the configuration file.

When you click the **Client** node, you can view or perform tasks on the client **Summary Page** in the **Detail Pane**.

Creating a new Client Endpoint Configuration

You can create a new client endpoint configuration in the following ways:

- Create by Wizard: Click the link **Create a New Client...** on the **Task Pane** on the lower-left of the window, or **Summary Page** to launch the wizard. You can also do so in the **File** menu -> **Add New Item**. The wizard prompts you to point to the location of the service configuration, from which the client configuration is generated. You can then choose the service endpoint to connect to.
- Create manually: Right-click the **Endpoints** node under **Client**, and choose **New Client Endpoint**.

Editing a Client Endpoint Configuration

1. Click a **Client Endpoint** node.
2. Edit the settings in the property grids.

Standard Endpoint

Standard endpoints are specialized endpoints that have one or more aspects of the address, contract and binding set to default values.

Such configuration settings are stored in the **Standard Endpoint** node. The **Standard Endpoint** node displays all of the standard endpoint settings in the configuration file. Every sub-node in the tree corresponds to a sub-element in the `<standardEndpoints>` element in the configuration file.

When you click the **Standard Endpoint** node, you can view or perform tasks on the standard endpoint

Summary Page in the **Detail Pane**.

Creating a New Standard Endpoint Configuration

You can create a new standard endpoint configuration in the following ways:

- Right-click the **Standard Endpoint** node and select **New Standard Endpoint Configuration...** Select the binding type in the dialog box and click **OK**.
- Select the **Standard Endpoint** node and click **New Standard Endpoint Configuration...** in the **Task Pane** on the lower-left of the window.

The **Creating a New Standard Endpoint** dialog box displays and lists all registered standard endpoint types.

Viewing and Editing a Standard Endpoint Configuration

You can open a standard endpoint configuration for viewing and editing in the following ways:

- Click to expand the **Standard Endpoint** node and click the respective endpoint sub-node.
- Click the **Standard Endpoint** node and click the respective endpoint on the Detail pane.

Attributes for the endpoint are shown in the right pane for editing.

Deleting a Standard Endpoint Configuration

You can delete a standard endpoint configuration in the following ways:

- Click to expand the **Standard Endpoint** node and right-click the respective endpoint sub-node. Use the context command **Delete Standard Endpoint Configuration** to delete the endpoint.
- Click the **Standard Endpoint** node. In the **Task** pane, click **Delete Standard Endpoint Configuration**.

If the standard endpoint is in used, a warning message is displayed when you attempt to delete it: **The standard endpoint is in use. If you delete it now, please be sure to delete all of its references in other parts of the configuration (for example, in the service endpoint or client endpoint). Otherwise, the configuration will be invalid and cannot be opened next time. Are you sure you want to delete the standard endpoint?"**

Binding

Binding configurations are used to configure bindings on endpoints. Such configuration settings are stored in the **Binding** node. Endpoints reference binding configurations by name and multiple endpoints can reference a single binding configuration.

The **Bindings** node displays all of the binding settings in the configuration file. Every sub-node in the tree corresponds to a sub-element in the `<bindings>` element in the configuration file.

When you click the **Bindings** node, you can view or perform tasks on the binding **Summary Page** in the **Detail Pane**.

Creating a New Binding Configuration

You can create a new binding configuration in the following ways.

- Right-click the **Bindings** node and select **New Binding Configuration...** Select the binding type in the dialog box and click **OK**.
- Select the **Bindings** node and click **New Binding Configuration...** in the **Task Pane** on the lower-left of the window.
- In the service or client summary page, click **Click to Create** in the **Binding Configuration** field to create a binding configuration for the corresponding endpoint.

Adding Binding Element Extensions to a Custom Binding

1. Select the binding you want to add an extension element to.

2. Click **Add**.
3. From the list of available extensions, select the binding element extension you want to add. To select multiple items, press the CTRL key simultaneously.
4. Click **Add**.

Adjusting the Extension Position in a Custom Binding

A custom binding is a collection of binding elements that form a stack. Each binding element on the stack has its own configuration settings. The order of the binding element extensions in a custom binding indicates their positions in the stack. Elements at the top of the stack are applied first. To change the ordering:

1. Select the custom binding node.
2. Select one binding extension element in the **Binding Element Extension Position** section.
3. Use the **Up** or **Down** button on the left side of the list to change the position of the selected element.

Editing the Configuration of Binding Element Extensions in a Custom Binding

1. Select the binding node in the tree.
2. Select the custom binding that contains the element you want to edit.
3. Select the binding element extension you want to edit. The settings of the element appear in the right pane, where they can be edited.

Diagnostics

The **Diagnostics** node displays all of the diagnostic settings in the configuration file. It enables you to turn performance counters on or off, enable or disable Windows Management Instrumentation (WMI), configure WCF tracing, and configure WCF message logging. The settings in the **Diagnostics** node correspond to the `<system.diagnostics>` section, and `<diagnostics>` section in `<system.serviceModel>` in the configuration file.

When you click the **Diagnostics** node, you can view or perform tasks on the diagnostics **Summary Page** in the **Detail Pane**.

Configuring performance counters and WMI

1. Click the **Diagnostics** node.
2. Click **Toggle Performance Counters**. The performance counter has three states: Off (default), ServiceOnly, and All. Clicking the link toggles the setting among these three states.

Configuring WMI Provider

1. Click the **Diagnostics** node.
2. To enable WMI provider, click the **Enable WMI Provider** link.

Enabling WCF Tracing

You can create a WCF trace file with standard properties or set up a custom trace file.

1. Click the **Diagnostics** node.
2. Click **Enable Tracing**.
3. Click the **Trace Level** link to adjust the trace level. There are six trace levels: Off, Critical, Error, Warning, Information, and Verbose. The **Activity Tracing** and **Propagate Activity** option enable you to use the WCF activity tracing feature.
4. Click the trace listener name to specify the trace file and options.

Enabling WCF Logging

You can create a WCF trace file with standard properties or set up a custom trace file.

1. Click the **Diagnostics** node.
2. Click **Enable Message Logging**.
3. Click the **Log Level** link to adjust the log level. There are three log levels: Malformed, Service, and Transport.
4. Click the listener name to specify the log file and options.

NOTE

If you want the trace and message logs to be flushed automatically when your application is closed, enable the **Auto Flush** option.

The **Diagnostics Summary Page** enables you to accomplish the most common tasks in configuring diagnostics. However, if you want to manually edit the Listeners and Sources settings, you must expand the **Diagnostics** node and edit settings in **Message Logging**, **Listeners** and **Sources** node.

Enabling WCF Custom Tracing or Message Logging

1. Click the **Diagnostics** node, and expand it.
2. Right-click the **Listeners** node and select **New Listener**.
3. Type in the trace file name in the **InitData** field. You can click the "..." button to browse to a path.
4. Clicking the **TypeName** line displays a "..." button. Click this button to open the **Trace Listener Type Browser**, which you can use to find pre-configured trace listeners that are already installed.
5. Note the **Source** section. Click **Add** in this section to open a dialog box with a drop-down menu, which lists available tracing sources. Select a tracing source and click **OK**.
6. To edit Message Logging settings, click the **Message Logging** node. You can edit the settings in the property grid.

Advanced

Behaviors

The **Behaviors** node displays the behaviors that are currently defined in the configuration file.

Behavior configurations are used to configure behaviors of endpoints and services. Such configuration settings are stored in the **Advanced** node under **Service Behaviors** and **Endpoint Behaviors**. Service behaviors are used by services; whereas endpoint behaviors by endpoints.

Behaviors are a collection of extension elements that for a stack. The element at the top of the stack is applied first. Each extension element can have its own configuration.

Creating a new Behavior Configuration

You can create a new behavior configuration in two ways.

- Right-click one of the behavior nodes and select "**New Behavior Configuration...**"
- Select one of the behavior nodes and click the **New Behavior Configuration...** in the **Task Pane** on the lower-left of the window.

Adding Behavior Element Extensions to a Behavior

1. Select one of the behavior nodes.
2. Select the behavior you want edit.
3. Click **Add**.
4. From the list of available extensions, select the behavior element extension you want to add.

5. Click **Add**.

Adjusting the Extension Position in a Behavior

Behaviors are collections of elements that form a stack. Each element on the stack has its own configuration. The order of the behavior element extensions in a behavior indicates their positions in the stack. Elements at the top of the stack are applied first. To change the ordering:

1. Select one of the behavior nodes.
2. Select the behavior you want edit.
3. Select a behavior extension element in the **Behavior Element Extension Position** section.
4. Use the **Up** or **Down** button on the left side of the list to change the position of the selected element.

Editing the Configuration of Behavior Element Extensions

1. Select one of the behavior nodes in the tree.
2. Select the behavior that contains the element you want to edit.
3. Select the behavior element extension you want to edit. The settings of the element appear in the right pane where they can be edited.

ProtocolMapping

This section allows you to set default binding types for different protocols such as http, tcp, MSMQ or net.pipe through defined mapping between protocol address schemes and the possible bindings. You can also add new mappings to other protocols.

Extensions

New binding extensions, binding element extensions, standard endpoint extensions and behavior extensions can be registered for use in WCF configuration. Extensions are name/type pairs. The name defines the name of the extension in configuration, whereas the type implements the extension. There are four types of extensions:

- Binding extensions define an entire binding type. Example: `basicHttpBinding`.
- Binding element extensions define an element of a binding. Example: `textMessageEncoding`.
- Standard endpoint extensions define an entire standard endpoint. Example: `discoveryEndpoint`.
- Behavior element extensions define an element of a behavior. Example: `clientVia`.

Extensions that have been registered in configuration can be used like any other WCF component of the same type.

Adding a new extension

Select one of the extension nodes in the advanced nodes:

1. Click **New**.
2. Enter a name and type.
3. Click **OK**.
4. The extension now appears in the appropriate place in the Editor. For example, if you add a behavior element extension, it appears in the list of available extensions.

Hosting Environment

This section allows you to define instantiation settings for the service hosting environment.

Creating a Configuration File Using the Wizard

One way to create a new configuration file is to use the New Service Element Wizard. The wizard finds the installed service types and other elements compatible with WCF on the computer, including COM+ and Web-

hosted virtual directories, and loads them to make creating the configuration much more streamlined.

Creating a Configuration File

1. Start Service Configuration Editor by using a command window to navigate to your WCF installation location, and then type `SvcConfigEditor.exe`.
2. From the **File** menu, select **Open** and click **Executable, COM + Service**, or **WebHosted Service**, depending on the type of configuration file you want to create.
3. In the **Open** dialog box, navigate to the specific file you want to create a configuration file for and double-click it.
4. In the **File** menu, point to **Add New Item** and click **Service**. The New Service Element Wizard opens.
5. Follow the steps in the wizard to create the new service.

NOTE

If you want to use the `NetPeerTcpBinding` from the configuration file generated by the Wizard, you have to manually add a binding configuration element and modify the `mode` attribute of its `security` element to "None".

Configuring COM+

The Service Configuration Editor enables you to create a new configuration file for an existing COM+ application, or edit an existing COM+ configuration. The **COM Contract** node is only visible when the `< comContract >` section exists in the configuration file.

Creating a New COM+ Configuration

Before creating a new COM+ configuration, make sure that your COM+ application is installed in Component Services, and registered in the Global Assembly Cache (GAC).

1. Select **File** menu -> **Integrate** -> **COM+ Application**. This operation closes the current opened file. If there is unsaved data in the current file, a Save dialog appears. The **COM+ Integration Wizard** is then launched.
2. In the first page, select the COM+ application from the tree. If you cannot find your COM+ application in the tree, verify that it is installed in the Component Services and registered in the Global Assembly Cache (GAC).
3. In the next page, select which method(s) you want to expose as WCF services. All the supported methods in the COM+ application are displayed and selected by default.
4. Choose a hosting method.
5. Configure other settings according to the guides in the wizard.
6. Service Configuration Editor utilizes `ComSvcConfig.exe` in the background to generate configuration file. After this is completed, you can view a summary and exit the wizard. The generated configuration file is opened so that you can edit it directly.

Editing an Existing COM+ Configuration

1. Select **File** menu -> **Open** -> **COM+ Service...**
2. Select the COM+ Service you want to edit from the list.
3. Edit configuration settings in the **COM Contracts** node.

NOTE

You can also directly open and edit a configuration file that contains COM contracts.

Security

A service configuration file generated by the Configuration Editor is not guaranteed to be secure. Please refer to the [Security](#) documentation to find out how to secure your WCF services.

In addition, the Configuration Editor can only be used to read and write valid WCF configuration elements. The tool ignores schema-compliant, user-defined elements. It also does not attempt remove these elements from the configuration file or determine their effects on the known WCF elements. It is the user's responsibility to determine whether these elements pose a threat to the application or the system.

COM+ Service Model Configuration Tool (ComSvcConfig.exe)

5/4/2018 • 3 minutes to read • [Edit Online](#)

The COM+ Service Model Configuration command-line tool (ComSvcConfig.exe) enables you to configure COM+ interfaces to be exposed as Web services.

Syntax

```
ComSvcConfig.exe /install | /uninstall | /list [/application:<ApplicationID | ApplicationName>] [/contract:  
<ClassID | ProgID | *,InterfaceID | InterfaceName | *>] [/hosting:<complus | was>] [/webSite:<WebsiteName>]  
[/webDirectory:<WebDirectoryName>] [/mex] [/id] [/nologo] [/verbose] [/help] [/partial]
```

Remarks

NOTE

You must be an administrator on the local computer to use ComSvcConfig.exe.

The tool can be found in the following location

%SystemRoot%\Microsoft.Net\Framework\v3.0\Windows Communication Foundation\

For more information about ComSvcConfig.exe, see [How to: Use the COM+ Service Model Configuration Tool](#).

The following table describes the modes that can be used with ComSvcConfig.exe.

OPTION	DESCRIPTION
<code>install</code>	Installs a configuration for a COM+ interface for Service Model integration. Short form <code>/i</code> .
<code>uninstall</code>	Uninstalls a configuration for a COM+ interface from Service Model integration. Short form <code>/u</code> .
<code>list</code>	Lists information about COM+ applications and components that have interfaces that are configured for Service Model integration. Short form <code>/l</code> .

The following table describes the flags that can be used with ComSvcConfig.exe.

OPTION	DESCRIPTION
--------	-------------

OPTION	DESCRIPTION
<code>/application:</code> <code><ApplicationID ApplicationName></code>	<p>Specifies the COM+ application to configure.</p> <p>Short form <code>/a</code>.</p>
<code>/contract:</code> <code><ClassID ProgID *,InterfaceID InterfaceName *></code>	<p>Specifies the COM+ component and interface that will be configured as the contract for the service.</p> <p>Short form <code>/c</code>.</p> <p>While the wildcard character (*) can be used when you specify the component and interface names, we recommend that you do not use it, because you might expose interfaces that you did not intend to.</p>
<code>/hosting:</code> <code><complus was></code>	<p>Specifies whether to use the COM+ hosting mode or the Web hosting mode.</p> <p>Short form <code>/h</code>.</p> <p>Using the COM+ hosting mode requires explicit activation of the COM+ application. Using the Web hosting mode allows the COM+ application to be automatically activated as required. If the COM+ application is a library application, it runs in the Internet Information Services (IIS) process. If the COM+ application is a server application, it runs in the Dllhost.exe process.</p>
<code>/webSite:</code> <code><WebsiteName></code>	<p>Specifies the Web site for hosting when Web hosting mode is used (see the <code>/hosting</code> flag).</p> <p>Short form <code>/w</code>.</p> <p>If no Web site is specified, the default Web site is used.</p>
<code>/webDirectory:</code> <code><WebDirectoryName></code>	<p>Specifies the virtual directory for hosting when Web hosting is used (see the <code>/hosting</code> flag).</p> <p>Short form <code>/d</code>.</p>
<code>/mex</code>	<p>Adds a Metadata Exchange (MEX) service endpoint to the default service configuration to support clients that want to retrieve a contract definition from the service.</p> <p>Short form <code>/x</code>.</p>
<code>/id</code>	<p>Displays the application, component, and interface information as IDs.</p> <p>Short form <code>/k</code>.</p>
<code>/nologo</code>	<p>Prevents ComSvcConfig.exe from displaying its logo.</p> <p>Short form <code>/n</code>.</p>

OPTION	DESCRIPTION
<code>/verbose</code>	<p>Outputs all warnings or informational text in addition to any errors encountered.</p> <p>Short form <code>/v</code>.</p>
<code>/help</code>	<p>Displays the usage message.</p> <p>Short form <code>/?</code>.</p>
<code>/partial</code>	<p>Generates a service configuration when the specified interface includes one or more method signatures that can be exposed. At service initialization time, compatible methods appear as operations on the service contract, and non-compatible methods are ignored and absent from the service contract.</p> <p>If this flag is missing, the tool will not generate a service configuration when the specified interface includes one or more incompatible methods.</p>

Examples

Description

The following example adds the `IFinances` interface of the `ItemOrders.Financial` component (from the OnlineStore COM+ application) to the set of interfaces that are exposed as Web services, using the COM+ hosting mode. All warnings will be output in addition to any errors encountered.

Code

```
ComSvcConfig.exe /install /application:OnlineStore /contract:ItemOrders.Financial,IFinances /hosting:complus /verbose
```

Description

The following example adds the `IStockLevels` interface of the `ItemInventory.Warehouse` component (from the OnlineWarehouse COM+ application) to the set of interfaces that are exposed as Web services, using the Web hosting mode. The Web service is Web hosted in the OnlineWarehouse virtual directory of IIS.

Code

```
ComSvcConfig.exe /install /application:OnlineWarehouse /contract:ItemInventory.Warehouse,IStockLevels /hosting:was /webDirectory:root/OnlineWarehouse
```

Description

The following example removes the `IFinances` interface of the `ItemOrders.Financial` component (from the OnlineStore COM+ application) from the set of interfaces that are exposed as Web services.

Code

```
ComSvcConfig.exe /uninstall /application:OnlineStore /interface:ItemOrders.Financial,IFinances /hosting:complus
```

Description

The following example lists currently exposed COM+ hosted interfaces, along with the corresponding address and binding details, for the OnlineStore COM+ application on the local machine.

Code

```
ComSvcConfig.exe /list /application:OnlineStore /hosting:complus
```

See Also

How to: [Use the COM+ Service Model Configuration Tool](#)

WS-AtomicTransaction Configuration Utility (wsatConfig.exe)

8/31/2018 • 2 minutes to read • [Edit Online](#)

The WS-AtomicTransaction Configuration Utility is used to configure basic WS-AtomicTransaction support settings.

Syntax

```
wsatConfig [Options]
```

Remarks

This command line tool can be used to configure basic WS-AT settings in a local machine only. If you have to configure settings on both local and remote machines, you should use the MMC snap-in as described in [Configuring WS-Atomic Transaction Support](#).

The command line tool can be found in the Windows SDK installation location, specifically,

%SystemRoot%\Microsoft.Net\Framework\v3.0\Windows Communication Foundation\wsatConfig.exe

If you are running Windows XP or Windows Server 2003, you must download an update before running WsatConfig.exe. For more information about this update, see [Update for Commerce Server 2007 \(KB912817\)](#) and [Availability of Windows XP COM+ Hotfix Rollup Package 13](#).

The following table shows the options that can be used with WS-AtomicTransaction Configuration Utility (wsatConfig.exe).

NOTE

When you set an SSL certificate for a selected port, you overwrite the original SSL certificate associated with that port if one exists.

OPTIONS	DESCRIPTION
-accounts:<account,>	Specifies a comma-separated list of accounts that can participate in WS-AtomicTransaction. The validity of these accounts is not checked.
-accountsCerts:<thumb> "Issuer\SubjectName",>	Specifies a comma-separated list of certificates that can participate in WS-AtomicTransaction. The certificates are indicated by thumbprint or by the Issuer\SubjectName pair. Use {EMPTY} for subject name if it is empty.
-endpointCert:<machine <thumb> "Issuer\SubjectName">	Uses the machine certificate or another local endpoint certificate specified by thumbprint or Issuer\SubjectName pair. Uses {EMPTY} for the subject name if it is empty.

OPTIONS	DESCRIPTION
-maxTimeout:<sec>	Specifies the maximum timeout in seconds. Valid values are from 0 to 3600.
-network:<enable disable>	Enables or disables the WS-AtomicTransaction network support.
-port:<portNum>	<p>Sets the HTTPS port for WS-AtomicTransaction.</p> <p>If you have already enabled firewall before running this tool, the port is automatically registered in the exception list. If firewall is disabled before running this tool, nothing additional is configured regarding the firewall.</p> <p>If you enable firewall after configuring WS-AT, you have to run this tool again and supply the port number using this parameter. If you disable firewall after configuring, WS-AT continues to work without additional input.</p>
-timeout:<sec>	Specifies the default timeout in seconds. Valid values are from 1 to 3600.
-traceActivity:<enable disable>	Enables or disables the tracing of activity events.
-traceLevel:<Off Error Critical Warning Information Verbose All>}	Specifies the trace level.
-tracePII:<enable disable>	Enables or disables the tracing of personally identifiable information.
-traceProp:<enable disable>	Enables or disables the tracing of propagation events.
-restart	Restarts MSDTC to activate changes immediately. If this is not specified, the changes take effect when MSDTC is restarted.
-show	Displays the current WS-AtomicTransaction protocol settings.
-virtualServer:<virtualServer>	Specifies the DTC resource cluster name.

See Also

[Using WS-AtomicTransaction](#)

[Configuring WS-Atomic Transaction Support](#)

Interpreting Error Codes Returned by wsatConfig.exe

5/4/2018 • 6 minutes to read • [Edit Online](#)

This topic lists all the error codes generated by the WS-AtomicTransaction Configuration Utility (wsatConfig.exe), and recommended actions to be taken.

List of Error Codes

ERROR CODE	DESCRIPTION	RECOMMENDED ACTION TO BE TAKEN
0	Operation was successful	None
1	Unexpected error	Contact Microsoft
2	An unexpected error occurred when trying to contact MSDTC to retrieve its security settings.	Ensure that the MSDTC service is not disabled, and address all problems listed in the returned Exception.
3	The account under which WsatConfig.exe ran did not have sufficient permissions to read the network security settings.	Execute WsatConfig.exe under an Administrator user account.
4	Enable "Network DTC Access" for MSDTC before trying to enable WS-AT support.	Enable "Network DTC Access" for MSDTC and re-run the utility.
5	The entered port was out of range. The value must be in the range of 1 to 65535.	Correct the <code>-port:<portNum></code> command line option as indicated in the error message.
6	An invalid endpoint certificate was specified on the command line. The certificate could not be found, or it did not pass verification.	Correct the <code>-endpointCert</code> command line option. Ensure that the certificate has a private key, is intended to be used for both ClientAuthentication and ServerAuthentication, is installed in the LocalMachine\MY certificate store, and is fully trusted.
7	An invalid accounts certificate was specified on the command line.	Correct the <code>-accountsCerts</code> command line option. The certificate specified was either improperly specified or it could not be found.
8	A default timeout was specified outside the range of 1 to 3600 seconds.	Enter a correct default timeout value as indicated.
10	An unexpected error occurred while trying to access the registry.	Check the error message and error code for actionable items

ERROR CODE	DESCRIPTION	RECOMMENDED ACTION TO BE TAKEN
11	Cannot write to the registry.	Ensure that the key listed in the error message is capable of supporting registry access from the account WsatConfig.exe was executed under.
12	An unexpected error occurred while trying to access the certificate store.	Use the error code returned to map to the appropriate system error.
13	Configuration of http.sys failed. Cannot create a new HTTPS port reservation for MSDTC.	Use the error code returned to map to the appropriate system error.
14	Configuration of http.sys failed. Cannot remove previous HTTPS port reservation for MSDTC.	Use the error code returned to map to the appropriate system error.
15	Configuration of http.sys failed. A previous HTTPS port reservation already exists for the specified port.	Another application has already taken ownership of the specific port. Change to a different port or uninstall or reconfigure the current application.
16	Configuration of http.sys failed. Cannot bind the specified certificate to the port.	Use the error code returned in the error message to map to the appropriate system error
17	Configuration of http.sys failed. Cannot unbind the SSL certificate from the previous port.	Use the error code returned in the error message to map to the appropriate system error. If necessary, use httpcfg.exe or netsh.exe to remove the erroneous port reservations.
18	Configuration of http.sys failed. Cannot bind the specified certificate to the port because a previous SSL binding already exists.	Another application has already taken ownership of the specific port. Change to a different port or uninstall or reconfigure the current application.
19	Restarting MSDTC failed	Manually restart MSDTC if necessary. If the problem persists, contact Microsoft.
20	WinFX is not installed on the remote machine, or is not installed correctly.	Install WinFX on the machine.
21	Remote configuration failed due to the operation timing out.	The call to configure WS-AT on the remote machine should take longer than 90 seconds.
22	WinFX is not installed on the remote machine, or is not installed correctly.	Install WinFX on the machine.
23	Remote configuration failed due to an exception on the remote machine.	Check the error message for actionable items
26	An invalid argument was passed to WsatConfig.exe.	Check the command line for errors.

ERROR CODE	DESCRIPTION	RECOMMENDED ACTION TO BE TAKEN
27	The <code>-accounts</code> command line option was invalid.	Correct the <code>-accounts</code> command line option to correctly specify a user account.
28	The <code>-network</code> command line option was invalid.	Correct the <code>-network</code> command line option to correctly specify "enable" or "disable".
29	The <code>-maxTimeout</code> command line option was invalid.	Correct the <code>-maxTimeout</code> command line option as indicated.
30	The <code>-timeout</code> command line option was invalid.	Correct the <code>-timeout</code> command line option as indicated.
31	The <code>-traceLevel</code> command line option was invalid.	Correct the <code>-traceLevel</code> command line option to specify a valid value from the followings, <ul style="list-style-type: none"> - Off - Error - Critical - Warning - Information - Verbose - All
32	The <code>-traceActivity</code> command line option was invalid.	Correct the <code>-traceActivity</code> command line option by specifying either "enable" or "disable".
33	The <code>-traceProp</code> command line option was invalid.	Correct the <code>-traceProp</code> command line option by specifying either "enable" or "disable".
34	The <code>-tracePII</code> command line option was invalid.	Correct the <code>-tracePII</code> command line option by specifying either "enable" or "disable".
37	WsatConfig.exe was not able to determine the exact machine certificate. This might happen when there is more than one candidate, or when none exists.	Specify a certificate thumbprint or Issuer\SubjectName pair to correctly identify the exact certificate to configure.
38	The process or user does not have sufficient permissions to change the firewall configuration.	Execute WsatConfig.exe under an Administrator user account.
39	WsatConfig.exe encountered an error while updating the firewall configuration.	Check the error message for actionable items.
40	WsatConfig.exe is not able to give MSDTC Read access to the certificate's private key file	Execute WsatConfig.exe under an Administrator user account.

ERROR CODE	DESCRIPTION	RECOMMENDED ACTION TO BE TAKEN
41	Either no installation of WinFX could be found, or the version found does not match what the tool is capable of configuring.	Ensure WinFX is correctly installed and only use the WsatConfig.exe tool that came with that version of WinFX to configure WS-AT.
42	An argument was specified more than once on the command line.	Only specify each argument once when executing WsatConfig.exe.
43	WsatConfig.exe cannot update WS-AT settings if WS-AT is not enabled.	Specify <code>-network:enable</code> as an additional command line argument.
44	A required hotfix is missing and WS-AT cannot be configured until the hotfix is installed.	See the WinFX release notes for instructions on installing the required hotfix.
45	The <code>-virtualServer</code> command line option was invalid.	Correct the <code>-virtualServer</code> command line option by specifying the network name of the cluster resource in which to configure.
46	An unexpected error occurred when trying to start the ETW trace session	Use the error code returned to map to the appropriate system error.
47	The process or user does not have sufficient permissions to enable the ETW trace session.	Execute WsatConfig.exe under an Administrator user account.
48	An unexpected error occurred while trying to start the ETW trace session.	Contact Microsoft.
49	Cannot create a new log file due to insufficient space on the %systemdrive%	Ensure that your %systemdrive% has adequate space for the log file.
51	An unexpected error occurred while trying to start the ETW trace session.	Contact Microsoft.
52	An unexpected error occurred while trying to start the ETW trace session.	Contact Microsoft.
53	Backup of the previous ETW session log file was unsuccessful.	Ensure that your %systemdrive% has adequate space for the log file and the backup of the previous log file (if any). Remove the previous log file manually if necessary.
55	An unexpected error occurred while trying to start the ETW trace session.	Contact Microsoft.
56	An unexpected error occurred while trying to start the ETW trace session.	Contact Microsoft.

See Also

[WS-AtomicTransaction Configuration Utility \(wsatConfig.exe\)](#)

WS-AtomicTransaction Configuration MMC Snap-in

5/4/2018 • 4 minutes to read • [Edit Online](#)

The WS-AtomicTransaction Configuration MMC Snap-in is used to configure a portion of the WS-AtomicTransaction settings on both local and remote machines.

Remarks

If you are running Windows XP or Windows Server 2003, the MMC snap-in can be found by navigating to **Control Panel/Administrative Tools/Component Services/**, right-clicking **My Computer**, and selecting **Properties**. This is the same location where you can configure the MSDTC. Options available for configuration are grouped under the **WS-AT** tab.

If you are running Windows Vista or Windows Server 2008, MMC snap-in can be found by clicking the **Start** button, and typing in `dcomcnfg.exe` in the **Search** box. When the MMC is opened, navigate to the **My Computer\Distributed Transaction Coordinator\Local DTC** node, right click and select **Properties**. Options available for configuration are grouped under the **WS-AT** tab.

The previous steps are used to launch the snap-in for configuring a local machine. If you want to configure a remote machine, you should locate the remote machine's name in **Control Panel/Administrative Tools/Component Services/**, and perform similar steps if you are running Windows XP or Windows Server 2003. If you are running Windows Vista or Windows Server 2008, follow the previous steps for Vista and Windows Server 2008, but use the **Distributed Transaction Coordinator\Local DTC** node under the remote computer's node.

To use the user interface provided by the tool, you have to register the WsatUI.dll file, which is located at the following path,

%PROGRAMFILES%\Microsoft SDKs\Windows\v6.0\Bin\WsatUI.dll

The registration can be done by the following command.

```
regasm.exe /codebase WsatUI.dll
```

You can use this tool to modify the basic WS-AtomicTransaction settings. For example, you can enable and disable the WS-AtomicTransaction protocol support, configure the HTTP ports for WS-AT, bind an SSL Certificate to the HTTP port, configure certificates by specifying certificate subject names, select the Tracing mode and set default and maximum timeouts.

If you must configure WS-AtomicTransaction support on the local machine only, you can use the command line version of this tool. For more information about the command line tool, see the [WS-AtomicTransaction Configuration Utility \(wsatConfig.exe\)](#) topic.

You should be aware that both the MMC Snap-in and the command-line tool do not support configuring all WS-AT settings. These settings can be edited only by modifying the registry directly. For more information about these registry settings, see [Configuring WS-Atomic Transaction Support](#).

User Interface Description

Enable WS-Atomic Transaction Network Support:

Toggling this checkbox enables or disables all the GUI components of this snap-in.

Before you check this box, you should make sure that Network DTC Access is enabled with inbound or outbound communication, or both. This value can be verified in the **Security** Tab of the MSDTC snap-in.

Network Group Box

You can specify the HTTPS port and additional security settings such as SSL encryption in the Network group. This group is disabled (grayed out) if DTC Network Transactions are not enabled.

HTTPS Port

This is the value of the HTTPS port used for WS-AT. The value must be a number in the range 1-65535 (as to represent a valid port). Changing the HTTP Port modifies the HTTP Service Configuration, which means that the previously used WS-AT Service Address is released, and a new WS-AT Service Address is registered based on the new port. In addition, the newly selected port is encrypted with the currently selected certificate for SSL Encryption.

NOTE

If you have already enabled the firewall before running this tool, the port is automatically registered in the exception list. If the firewall is disabled before running this tool, nothing additional is configured regarding the firewall.

If you enable the firewall after configuring WS-AT, you must run this tool again and supply the port number using this parameter. If you disable the firewall after configuring, WS-AT continues to work without additional input.

Endpoint Certificate

Clicking the **Select** button displays a list with the currently available certificates on the Local Machine, allowing the user to select the certificate that can be used for SSL encryption. The certificates must have a private key. Otherwise, you receive an error message.

NOTE

When you set an SSL certificate for a selected port, you overwrite the original SSL certificate associated with that port if one exists.

Authorized Accounts

Clicking the **Select** button invokes the Windows Access Control List editor, where you can specify the user or group that can participate in WS-Atomic transactions by checking the **Allow** or **Deny** box in the **Participate** permission group.

Authorized Certificates

Clicking the **Select** button displays a list of currently available certificates on the LocalMachine. You can then select which certificate identities are allowed to participate in WS-Atomic transactions.

Timeout Group Box

The **Timeout** group box allows you to specify the default and maximum timeout for a WS-Atomic transaction. A valid value for outgoing timeout is between 1 and 3600. A valid value for incoming timeout is between 0 and 3600.

Tracing and Logging Group Box

The **Tracing and Logging** group Box allows you to configure the desired tracing and logging level.

Clicking the **Options** button invokes a page where you can specify additional settings.

The **Trace Level** combination box allows you to choose from any valid value of the [TraceLevel](#) enumeration. You can also use the checkboxes to specify if you want to perform activity tracing, activity propagation or collect personal identifiable information.

You can also specify logging sessions in the **Logging Session** group box.

NOTE

When another trace consumer is using the WS-AT trace provider, you cannot create a new logging session for trace events. Any attempt to configure logging during this time results in the error message "Failed to enable provider. Error code: 1".

For more information about tracing and logging, see [Administration and Diagnostics](#).

See Also

[Configuring WS-Atomic Transaction Support](#)

[WS-AtomicTransaction Configuration Utility \(wsatConfig.exe\)](#)

[Administration and Diagnostics](#)

WorkFlow Service Registration Tool (WFServicesReg.exe)

5/4/2018 • 2 minutes to read • [Edit Online](#)

Workflow Services Registration tool (WFServicesReg.exe) is a stand-alone tool that can be used to add, remove, or repair the configuration elements for Windows Workflow Foundation (WF) services.

Syntax

```
WFServicesReg.exe [-c | -r | -v | -m | -i]
```

Remarks

The tool can be found at the .NET Framework 3.5 installation location, specifically, %windir%\Microsoft.NET\Framework\v3.5, or at %windir%\Microsoft.NET\Framework64\v3.5 in 64-bit machines.

The following tables describe the options that can be used with the Workflow Services Registration tool (WFServicesReg.exe).

OPTION	DESCRIPTION
<code>/c</code>	Configures Windows Workflow Services. Used in install and repair scenarios.
<code>/r</code>	Removes Windows Workflow Services Configuration.
<code>/v</code>	Print verbose information (for either configuration or removal).
<code>/m</code>	Enables MSI logging format.
<code>/i</code>	Minimizes the window when the application runs.

Registration

The tool inspects the Web.config file and registers the following:

- .NET Framework 3.5 reference assemblies.
- A build provider for .xoml files.
- HTTP handlers for .xoml and .rules files.

The tool inspects the Machine.config file and registers the following extensions:

- behaviorExtensions
- bindingElementExtensions

- bindingExtensions

The tool also registers the following client metadata importers:

- policyImporters
- wsdlImporters

The tool also registers .xml and .rules scriptmaps and handlers in the IIS metabase.

On Windows Server 2003 and Windows XP machines (IIS 5.1 and IIS 6.0), one set of .xml and .rules scriptmaps are registered.

On 64-bit machines, the tool registers WOW mode scriptmaps if the `Enable32BitAppOnWin64` switch is enabled, or native 64-bit scriptmaps if the `Enable32BitAppOnWin64` switch is disabled.

On Windows Vista and Windows Server 2008 (IIS 7.0 and above) machines, two sets of .xml and .rules handlers are registered: one for Integrated mode and one for Classic mode.

On 64-bit machines, three sets of handlers are registered (regardless of the state of the `Enable32BitAppOnWin64` switch): one for Integrated mode, one for WOW Classic mode and one for native 64-bit Classic mode.

NOTE

Unlike ServiceModelreg.exe, WFServicesReg.exe does not allow adding, removing, or repairing scriptmaps or handlers for a particular Web site. For a workaround to this issue, see the "Repairing the Scriptmaps" section.

Usage Scenarios

Installing IIS after .NET Framework 3.5 is installed

On a Windows Server 2003 machine, .NET Framework 3.5 is installed prior to IIS installation. Due to the unavailability of the IIS metabase, installation of .NET Framework 3.5 succeeds without installing .xml and .rules scriptmaps.

After IIS is installed, you can use the WFServicesReg.exe tool with the `/c` switch to install these specific scriptmaps.

Repairing the Scriptmaps

Scriptmap deleted under Web Sites node

On a Windows Server 2003 machine, .xml or .rules is accidentally deleted from the Web Sites node. This can be repaired by running the WFServicesReg.exe tool with the `/c` switch.

Scriptmap deleted under a particular Web site

On a Windows Server 2003 machine, .xml or .rules is accidentally deleted from a particular Web site (for example, the Default Web Site) rather than from the Web Sites node.

To repair deleted handlers for a particular Web site, you should run "WFServicesReg.exe /r" to remove handlers from all Web sites, then run "WFServicesReg.exe /c" to create the appropriate handlers for all Web sites.

Configuring handlers after switching IIS mode

When IIS is in shared configuration mode and .NET Framework 3.5 is installed, the IIS metabase is configured under a shared location. If you switch IIS to non-shared configuration mode, the local metabase will not contain the required handlers. To configure the local metabase properly, you can either import the shared metabase to local, or run "WFServicesReg.exe /c", which configures the local metabase.

Contract-First Tool

10/3/2018 • 6 minutes to read • [Edit Online](#)

Service contracts often need to be created from existing services. In .NET Framework 4.5, data contract classes can be created automatically from existing services using the contract-first tool. To use the contract-first tool, the XML schema definition file (XSD) must be downloaded locally; the tool cannot import remote data contracts via HTTP.

The contract-first tool is integrated into Visual Studio 2012 as a build task. The code files generated by the build task are created every time the project is built, so that the project can easily adopt changes in the underlying service contract.

Schema types that the contract-first tool can import include the following:

```
<xsd:complexType>
<xsd:simpleType>
```

Simple types will not be generated if they are primitives such as `Int16` or `String`; complex types will not be generated if they are of type `Collection`. Types will also not be generated if they are part of another `xsd:complexType`. In all these cases, the types will be referenced to existing types in the project instead.

Adding a data contract to a project

Before the contract-first tool can be used, the service contract (XSD) must be added to the project. For the purposes of this overview, the following contract will be used to illustrate contract-first functions. This service definition is a small subset of the service contract used by Bing's search API.

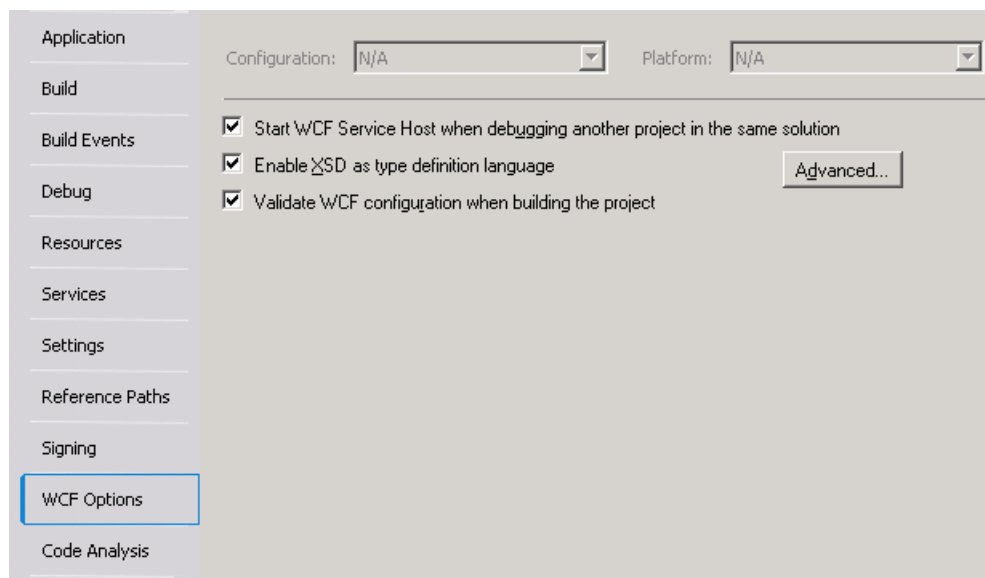
```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="ServiceSchema"
  targetNamespace="http://tempuri.org/ServiceSchema.xsd"
  elementFormDefault="qualified"
  xmlns="http://tempuri.org/ServiceSchema.xsd"
  xmlns:mstns="http://tempuri.org/ServiceSchema.xsd"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
  <xs:complexType name="SearchRequest">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="1" name="Version" type="xs:string" default="2.2" />
      <xs:element minOccurs="0" maxOccurs="1" name="Market" type="xs:string" />
      <xs:element minOccurs="0" maxOccurs="1" name="UILanguage" type="xs:string" />
      <xs:element minOccurs="1" maxOccurs="1" name="Query" type="xs:string" />
      <xs:element minOccurs="1" maxOccurs="1" name="AppId" type="xs:string" />
      <xs:element minOccurs="0" maxOccurs="1" name="Latitude" type="xs:double" />
      <xs:element minOccurs="0" maxOccurs="1" name="Longitude" type="xs:double" />
      <xs:element minOccurs="0" maxOccurs="1" name="Radius" type="xs:double" />
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="WebSearchOption">
    <xs:restriction base="xs:string">
      <xs:enumeration value="DisableHostCollapsing" />
      <xs:enumeration value="DisableQueryAlterations" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

To add the above service contract to the project, right-click the project and select **Add New....** Select Schema

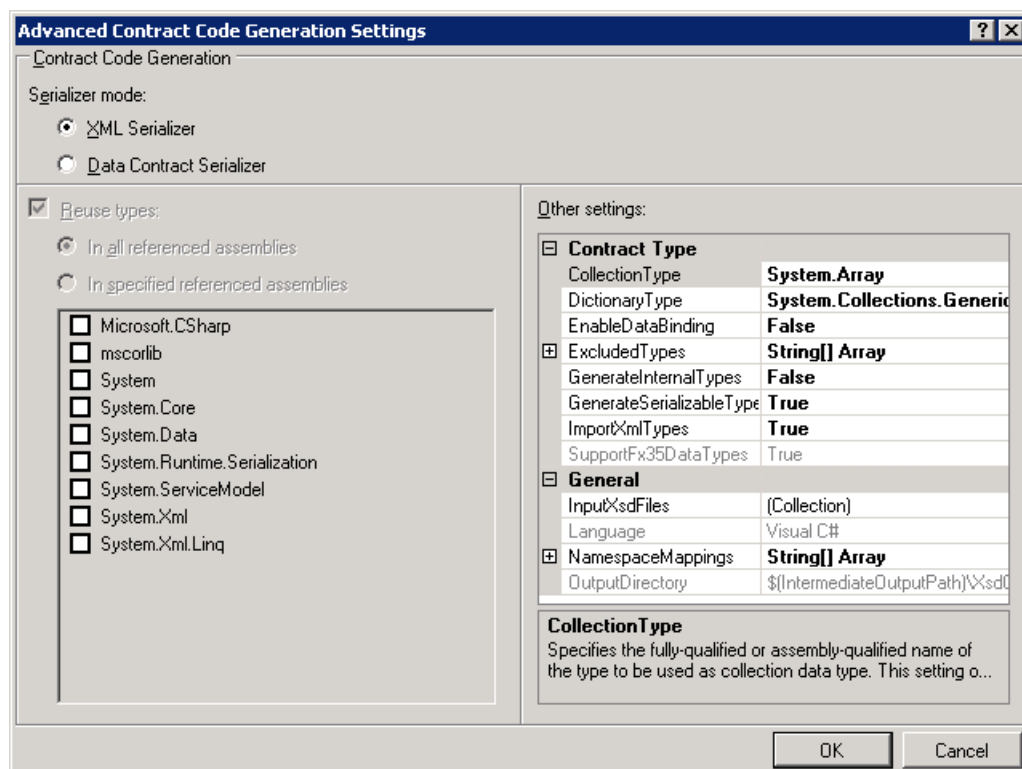
Definition from the WCF pane of the Templates dialog, and name the new file SampleContract.xsd. Copy and paste the above code into the code view of the new file.

Configuring contract-first options

Contract-first options can be configured in the Properties menu of a WCF project. To enable contract-first development, select the **Enable XSD as Type Definition Language** check box in the WCF page of the project properties window.



To configure advanced properties, click the Advanced button.



The following advanced settings can be configured for code generation from contracts. Settings can only be configured for all of the files in the project; settings cannot be configured for individual files at this time.

- **Serializer Mode:** This setting determines which serializer is used for reading service contract files. When **XML Serializer** is selected, the **Collection Types** and **Reuse Types** options are disabled. These options only apply to the **Data Contract Serializer**.
- **Reuse Types:** This setting specifies which libraries are used for type reuse. This setting only applies if

Serializer Mode is set to **Data Contract Serializer**.

- **Collection Type:** This setting specifies the fully-qualified or assembly-qualified type to be used for the collection data type. This setting only applies if **Serializer Mode** is set to **Data Contract Serializer**.
- **Dictionary Type:** This setting specifies the fully-qualified or assembly-qualified type to be used for the dictionary data type.
- **EnableDataBinding:** This setting specifies whether to implement the [INotifyPropertyChanged](#) interface on all data types to implement data binding.
- **ExcludedTypes:** This setting specifies the list of fully-qualified or assembly-qualified types to be excluded from the referenced assemblies. This setting only applies if **Serializer Mode** is set to **Data Contract Serializer**.
- **GenerateInternalTypes:** This setting specifies whether to generate classes that are marked as internal. This setting only applies if **Serializer Mode** is set to **Data Contract Serializer**.
- **GenerateSerializableTypes:** This setting specifies whether to generate classes with the [SerializableAttribute](#) attribute. This setting only applies if **Serializer Mode** is set to **Data Contract Serializer**.
- **ImportXMLTypes:** This setting specifies whether to configure the data contract serializer to apply the [SerializableAttribute](#) attribute to classes without the [DataContractAttribute](#) attribute. This setting only applies if **Serializer Mode** is set to **Data Contract Serializer**.
- **SupportFx35TypedDataSets:** This setting specifies whether to provide additional functionality for typed data sets created for .Net Framework 3.5. When **Serializer Mode** is set to **XML Serializer**, the [TypedDataSetSchemaImporterExtensionFx35](#) extension will be added to the XML schema importer when this value is set to True. When **Serializer Mode** is set to **Data Contract Serializer**, the type [DateTimeOffset](#) will be excluded from the References when this value is set to False, so that a [DateTimeOffset](#) is always generated for older framework versions.
- **InputXsdFiles:** This setting specifies the list of input files. Each file must contain a valid XML schema.
- **Language:** This setting specifies the language of the generated contract code. The setting must be recognizable by [CodeDomProvider](#).
- **NamespaceMappings:** This setting specifies the mappings from the XSD Target Namespaces to CLR namespaces. Each mapping should use the following format:

```
"<Schema Namespace>, <CLR Namespace>"
```

The XML Serializer only accepts one mapping in the following format:

```
"*, <CLR Namespace>"
```

- **OutputDirectory:** This setting specifies the directory where the code files will be generated.

The settings will be used to generate service contract types from the service contract files when the project is built.

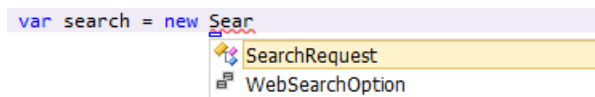
Using contract-first development

After adding the service contract to the project and confirming the build settings, build the project by pressing **F6**. The types defined in the service contract will then be available for use in the project.

To use the types defined in the service contract, add a reference to `ContractTypes` under the current namespace:

```
using MyProjectNamespace.ContractTypes;
```

The types defined in the service contract will then be resolvable in the project, as shown below.



The types generated by the tool are created in the GeneratedXSDTypes.cs file. The file is created in the <project directory>/obj/<build configuration>/XSDGeneratedCode/ directory by default. The sample schema at the beginning of this topic is converted as follows:

```
//-----  
// <auto-generated>  
//   This code was generated by a tool.  
//   Runtime Version:4.0.30319.17330  
//  
//   Changes to this file may cause incorrect behavior and will be lost if  
//   the code is regenerated.  
// </auto-generated>  
//-----  
  
namespace TestXSD3.ContractTypes  
{  
    using System.Xml.Serialization;  
  
    /// <remarks/>  
    [System.CodeDom.Compiler.GeneratedCodeAttribute("System.Xml", "4.0.30319.17330")]  
    [System.SerializableAttribute()]  
    [System.Diagnostics.DebuggerStepThroughAttribute()]  
    [System.ComponentModel.DesignerCategoryAttribute("code")]  
    [System.Xml.Serialization.XmlTypeAttribute(Namespace="http://tempuri.org/ServiceSchema.xsd")]  
    [System.Xml.Serialization.XmlRootAttribute(Namespace="http://tempuri.org/ServiceSchema.xsd",  
        IsNullable=true)]  
    public partial class SearchRequest  
    {  
  
        private string versionField;  
  
        private string marketField;  
  
        private string uILanguageField;  
  
        private string queryField;  
  
        private string appIdField;  
  
        private double latitudeField;  
  
        private bool latitudeFieldSpecified;  
  
        private double longitudeField;  
  
        private bool longitudeFieldSpecified;  
  
        private double radiusField;  
  
        private bool radiusFieldSpecified;  
  
        public SearchRequest()  
        {  
            this.versionField = "2.2";  
        }  
  
        /// <remarks/>  
    }  
}
```

```

/// <remarks/>
[System.ComponentModel.DefaultValueAttribute("2.2")]
public string Version
{
    get
    {
        return this.versionField;
    }
    set
    {
        this.versionField = value;
    }
}

/// <remarks/>
public string Market
{
    get
    {
        return this.marketField;
    }
    set
    {
        this.marketField = value;
    }
}

/// <remarks/>
public string UILanguage
{
    get
    {
        return this.uILanguageField;
    }
    set
    {
        this.uILanguageField = value;
    }
}

/// <remarks/>
public string Query
{
    get
    {
        return this.queryField;
    }
    set
    {
        this.queryField = value;
    }
}

/// <remarks/>
public string AppId
{
    get
    {
        return this.appIdField;
    }
    set
    {
        this.appIdField = value;
    }
}

/// <remarks/>
public double Latitude
{
    get

```

```

        get
        {
            return this.latitudeField;
        }
        set
        {
            this.latitudeField = value;
        }
    }

    /// <remarks/>
    [System.Xml.Serialization.XmlIgnoreAttribute()]
    public bool LatitudeSpecified
    {
        get
        {
            return this.latitudeFieldSpecified;
        }
        set
        {
            this.latitudeFieldSpecified = value;
        }
    }

    /// <remarks/>
    public double Longitude
    {
        get
        {
            return this.longitudeField;
        }
        set
        {
            this.longitudeField = value;
        }
    }

    /// <remarks/>
    [System.Xml.Serialization.XmlIgnoreAttribute()]
    public bool LongitudeSpecified
    {
        get
        {
            return this.longitudeFieldSpecified;
        }
        set
        {
            this.longitudeFieldSpecified = value;
        }
    }

    /// <remarks/>
    public double Radius
    {
        get
        {
            return this.radiusField;
        }
        set
        {
            this.radiusField = value;
        }
    }

    /// <remarks/>
    [System.Xml.Serialization.XmlIgnoreAttribute()]
    public bool RadiusSpecified
    {
        get
        {

```

```

        {
            return this.radiusFieldSpecified;
        }
        set
        {
            this.radiusFieldSpecified = value;
        }
    }
}

/// <remarks/>
[System.CodeDom.Compiler.GeneratedCodeAttribute("System.Xml", "4.0.30319.17330")]
[System.SerializableAttribute()]
[System.Xml.Serialization.XmlTypeAttribute(Namespace="http://tempuri.org/ServiceSchema.xsd")]
[System.Xml.Serialization.XmlRootAttribute(Namespace="http://tempuri.org/ServiceSchema.xsd",
IsNullable=false)]
public enum WebSearchOption
{

    /// <remarks/>
    DisableHostCollapsing,

    /// <remarks/>
    DisableQueryAlterations,

}
}

```

Errors and warnings

Errors and warnings encountered in parsing the XSD schema will appear as build errors and warnings.

Interface Inheritance

It is not possible to use interface inheritance with contract-first development; this is consistent with the way interfaces behave in other operations. In order to use an interface that inherits a base interface, use two separate endpoints. The first endpoint uses the inherited contract, and the second endpoint implements the base interface.

Windows Communication Foundation (WCF) samples

10/19/2018 • 2 minutes to read • [Edit Online](#)

You can download [Windows Communication Foundation \(WCF\) samples for .NET Framework 4](#). The samples provide instruction on various aspects of Windows Communication Foundation (WCF).

TIP

The articles in this section describe some of the samples in the download package. For a complete documentation set that covers all of the samples, check the [.NET Framework 4 documentation for WCF samples](#).

The Windows Workflow Foundation (WF) [application](#) samples also demonstrate several WCF features.

In this section

[Basic](#) - Samples that illustrate basic WCF functionality.

[Extensibility](#) - Samples that are related to the discovery feature.

[Scenario](#) - Demonstrates a WCF scenario.

Windows Communication Foundation Glossary for .NET Framework 4.5

5/4/2018 • 10 minutes to read • [Edit Online](#)

The following terms are defined for Windows Communication Foundation documentation.

Terms

TERM	DEFINITION
address	Specifies the location where messages are received. It is specified as a Uniform Resource Identifier (URI). The URI schema part names the transport mechanism to use to reach the address, such as HTTP and TCP. The hierarchical part of the URI contains a unique location whose format is dependent on the transport mechanism.
application endpoint	An endpoint exposed by the application and that corresponds to a service contract implemented by the application.
behavior	A behavior is a component that controls various run-time aspects of a service, an endpoint, a particular operation, or a client. Behaviors are grouped according to scope: common behaviors affect all endpoints globally, service behaviors affect only service-related aspects, endpoint behaviors affect only endpoint-related properties, and operation-level behaviors affect particular operations.
binding	Defines which communication protocols are used to communicate with WCF services. It is constructed of a set of components called binding elements that stack one on top of the other to create the communication infrastructure. See endpoint.
channel	A concrete implementation of a binding element. The binding represents the configuration, and the channel is the implementation associated with that configuration. Therefore, there is a channel associated with each binding element. Channels stack on top of each other to create the concrete implementation of the binding: the channel stack.
claims-based security	Allows authorized access to protected resources based on claims.
client application	A client application is a program that exchanges messages with one or more endpoints. The client application begins by creating an instance of a WCF client and calling methods of the WCF client. It is important to note that a single application can be both a client and a service.

TERM	DEFINITION
coding	Allows the developer to retain strict control over all components of the service or client, and any settings done through the configuration can be inspected and if needed overridden by the code. Control of an application can be done either through coding, through configuration, or through a combination of both.
configuration	Configuration has the advantage of allowing someone other than the developer (for example, a network administrator) to set client and service parameters after the code is written and without having to recompile. Configuration not only enables you to set values like endpoint addresses, but also allows further control by enabling you to add endpoints, bindings, and behaviors. Control of an application can be done either through configuration, through coding, or through a combination of both.
contract	A contract is a specification of support for the particular type of contract that it is. A service contract, for example, is a specification for a group of operations. In WCF, contracts have a hierarchy that is mirrored in the description objects located in the System.ServiceModel.Description namespace. A service contract is the largest contract scope in WCF. Each service operation in a service contract has an operation contract, which specifies the messages -- including fault messages -- the operation can exchange, and in which direction. Each message in an operation has a message contract, a specification for the structure of the SOAP message envelope, and each message contract has a data contract, which specifies the data structures contained in the messages.
data contract	The data types a service uses must be described in metadata to enable others to interoperate with the service. The descriptions of the data types are known as the data contract, and the types can be used in any part of a message, for example, as parameters or return types. If the service is using only simple types, there is no need to explicitly use data contracts.
declarative application	An application that is described sufficiently to be created at runtime without running imperative instructions.
endpoint	Consists of an address, a binding, and a contract used for communicating with a WCF service.
endpoint address	Enables you to create unique endpoint addresses for each endpoint in a service, or under certain conditions share an address across endpoints.
fault contract	A fault contract can be associated with a service operation to denote errors that can be returned to the caller. An operation can have zero or more faults associated with it. These errors are SOAP faults that are modeled as exceptions in the programming model. The exception is converted into a SOAP fault that can then be sent to the client.

TERM	DEFINITION
hosting	A service must be hosted in some process. A host is an application that controls the lifetime of the service. Services can be self-hosted or managed by an existing hosting process.
hosting process	A hosting process is an application that is designed to host services. These include Internet Information Services (IIS), Windows Activation Services (WAS), and Windows Services. In these hosted scenarios, the host controls the lifetime of the service. For example, using IIS you can set up a virtual directory that contains the service assembly and configuration file. When a message is received, IIS starts the service and controls its lifetime.
initiating operation	An operation that is called as the first operation of a new session. Non-initiating operations can be called only after at least one initiating operation has been called.
instancing model	A service has an instancing model. There are three instancing models: "single," in which a single CLR object services all the clients; "per call," in which a new CLR object is created to handle each client call; and "per session," in which a set of CLR objects are created, one for each separate session. The choice of an instancing model depends on the application requirements and the expected usage pattern of the service.
message	A message is a self-contained unit of data that may consist of several parts, including a body and headers.
message contract	A message contract describes the format of a message. For example, it declares whether message elements should go in headers versus the body, what level of security should be applied to what elements of the message, and so on.
message security mode	Message security mode specifies that security is provided by implementing one or more of the security specifications. Each message contains the necessary mechanisms to provide security during its transit, and to enable the receivers to detect tampering and to decrypt the messages. In this sense, the security is encapsulated within every message, providing end-to-end security across multiple hops. Because security information becomes part of the message, it is also possible to include multiple kinds of credentials with the message (these are referred to as claims). This approach also has the advantage of enabling the message to travel securely over any transport, including multiple transports between its origin and destination. The disadvantage of this approach is the complexity of the cryptographic mechanisms employed, resulting in performance implications.

TERM	DEFINITION
metadata	The metadata of a service describes the characteristics of the service that an external entity needs to understand to communicate with the service. Metadata can be consumed by the ServiceModel Metadata Utility Tool (Svcutil.exe) to generate a WCF client and accompanying configuration that a client application can use to interact with the service. The metadata exposed by the service includes XML schema documents, which define the data contract of the service, and WSDL documents, which describe the methods of the service. When enabled, metadata for the service is automatically generated by WCF by inspecting the service and its endpoints. To publish metadata from a service, you must explicitly enable the metadata behavior.
operation contract	An operation contract defines the parameters and return type of an operation. When creating an interface that defines the service contract, you signify an operation contract by applying the <code>T:System.ServiceModel.OperationContractAttribute</code> attribute to each method definition that is part of the contract. The operations can be modeled as taking a single message and returning a single message, or as taking a set of types and returning a type. In the latter case, the system determines the format for the messages that are exchanged for that operation.
projection	The representation of data on the wire. For example, a SOAP projection sends messages as SOAP envelopes and a Web projection sends messages in JSON format.
security	Security in WCF includes confidentiality (encryption of messages to prevent eavesdropping), integrity (the means for detection of tampering with the message), authentication (the means for validation of servers and clients), and authorization (the control of access to resources). These functions are provided by either leveraging existing security mechanisms, such as TLS over HTTP (also known as HTTPS), or by implementing one or more of the various WS-* security specifications.
self-hosted service	A self-hosted service is one that runs within a process application that the developer created. The developer controls its lifetime, sets the properties of the service, opens the service (which sets it into a listening mode), and closes the service.
service	A program or process that exposes one or more endpoints, with each endpoint exposing one or more operations.
service contract	The service contract ties together multiple related operations into a single functional unit. The contract can define service-level settings, such as the namespace of the service, a corresponding callback contract, and other such settings. In most cases, the contract is defined by creating an interface in the programming language of your choice and applying the <code>T:System.ServiceModel.ServiceContractAttribute</code> attribute to the interface. The actual service code results by implementing the interface.

TERM	DEFINITION
service operation	A service operation is a procedure defined in a service's code that implements the functionality for an operation. This operation is exposed to clients as methods on a WCF client. The method may return a value, and may take an optional number of arguments, or take no arguments, and return no response. For example, an operation that functions as a "Hello" can be used as a notification of a client's presence and to begin a series of operations.
system-provided bindings	WCF includes a number of system-provided bindings. These are collections of binding elements that are optimized for specific scenarios. For example, the <code>T:System.ServiceModel.WSHttpBinding</code> is designed for interoperability with services that implement various WS-* specifications. These bindings save time by presenting only those options that can be correctly applied to the specific scenario. If one of these bindings does not meet your requirements, you can create your own custom binding.
terminating operation	An operation that is called as the last message in an existing session. In the default case, WCF recycles the service object and its context after the session with which the service was associated is closed.
transport security mode	Security can be provided by one of three modes: transport mode, message security mode, and transport with message credential mode. The transport security mode specifies that confidentiality, integrity, and authentication are provided by the transport layer mechanisms (such as HTTPS). When using a transport like HTTPS, this mode has the advantage of being efficient in its performance, and well understood because of its prevalence on the Internet. The disadvantage is that this kind of security is applied separately on each hop in the communication path, making the communication susceptible to a "man in the middle" attack.
transport with message credential security mode	This mode uses the transport layer to provide confidentiality, authentication, and integrity of the messages, while each of the messages can contain multiple credentials (claims) required by the receivers of the message.
type converter	A CLR type can be associated with one or more <code>System.ComponentModel.TypeConverter</code> derived types that enable converting instances of the CLR type to and from instances of other types. A type converter is associated with a CLR type using the <code>System.ComponentModel.TypeConverterAttribute</code> attribute. A <code>TypeConverterAttribute</code> can be specified directly on the CLR type or on a property. A type converter specified on a property always takes precedence over a type converter specified on the CLR type of the property.

TERM	DEFINITION
WCF client	A WCF client is a client-application construct that exposes the service operations as methods (in the .NET Framework programming language of your choice, such as Visual Basic or Visual C#). Any application can host a WCF client, including an application that hosts a service. Therefore, it is possible to create a service that includes WCF clients of other services. A WCF client can be automatically generated by using the ServiceModel Metadata Utility Tool (Svcutil.exe) and pointing it at a running service that publishes metadata.
workflow services	A workflow service is a WCF service that is implemented as a workflow. The workflow contains messaging activities that send and/or receive WCF messages.
WS-*	Shorthand for the growing set of Web Service (WS) specifications, such as WS-Security, WS-ReliableMessaging, and so on, that are implemented in WCF.
XAML	eXtensible Application Markup Language
XAML schema	A markup schema used to define custom types in XAML.

General Reference

5/4/2018 • 2 minutes to read • [Edit Online](#)

The section `<system.serviceModel>` describes the elements that are used to configure Windows Communication Foundation (WCF) clients and services.

Feedback and Community

8/31/2018 • 2 minutes to read • [Edit Online](#)

We appreciate your comments and concerns about the Windows Communication Foundation (WCF) or CardSpace documentation. On the bottom of every page in the SDK is the sentence "Send comments about this topic to Microsoft." Click the "comments" link to send your views or suggestions about the specific topic.

If you are looking for more information, or want to share techniques with others who are interested in WCF, try the [MSDN Windows Communication Foundation forum](#).

Windows Communication Foundation Privacy Information

10/5/2018 • 13 minutes to read • [Edit Online](#)

Microsoft is committed to protecting end-users' privacy. When you build an application using Windows Communication Foundation (WCF), version 3.0, your application may impact your end-users' privacy. For example, your application may explicitly collect user contact information, or it may request or send information over the Internet to your Web site. If you embed Microsoft technology in your application, that technology may have its own behavior that might affect privacy. WCF does not send any information to Microsoft from your application unless you or the end-user choose to send it to us.

WCF in Brief

WCF is a distributed messaging framework using the Microsoft .NET Framework that allows developers to build distributed applications. Messages communicated between two applications contain header and body information.

Headers may contain message routing, security information, transactions, and more depending on the services used by the application. Messages are typically encrypted by default. The one exception is when using the `BasicHttpBinding`, which was designed for use with non-secured, legacy Web services. As the application designer, you are responsible for the final design. Messages in the SOAP body contain application-specific data; however, this data, such as application-defined personal information, can be secured by using WCF encryption or confidentiality features. The following sections describe the features that potentially impact privacy.

Messaging

Each WCF message has an address header that specifies the message destination and where the reply should go.

The address component of an endpoint address is a Uniform Resource Identifier (URI) that identifies the endpoint. The address can be a network address or a logical address. The address may include machine name (hostname, fully qualified domain name) and an IP address. The endpoint address may also contain a globally unique identifier (GUID), or a collection of GUIDs for temporary addressing used to discern each address. Each message contains a message ID that is a GUID. This feature follows the WS-Addressing reference standard.

The WCF messaging layer does not write any personal information to the local machine. However, it might propagate personal information at the network level if a service developer has created a service that exposes such information (for example, by using a person's name in an endpoint name, or including personal information in the endpoint's Web Services Description Language but not requiring clients to use https to access the WSDL). Also, if a developer runs the [ServiceModel Metadata Utility Tool \(Svcutil.exe\)](#) tool against an endpoint that exposes personal information, the tool's output could contain that information, and the output file is written to the local hard disk.

Hosting

The hosting feature in WCF allows applications to start on demand or to enable port sharing between multiple applications. An WCF application can be hosted in Internet Information Services (IIS), similar to ASP.NET.

Hosting does not expose any specific information on the network and it does not keep data on the machine.

Message Security

WCF security provides the security capabilities for messaging applications. The security functions provided include

authentication and authorization.

Authentication is performed by passing credentials between the clients and services. Authentication can be either through transport-level security or through SOAP message-level security, as follows:

- In SOAP message security, authentication is performed through credentials like username/passwords, X.509 certificates, Kerberos tickets, and SAML tokens, all of which might contain personal information, depending on the issuer.
- Using transport security, authentication is done through traditional transport authentication mechanisms like HTTP authentication schemes (Basic, Digest, Negotiate, Integrated Windows Authorization, NTLM, None, and Anonymous), and form authentication.

Authentication can result in a secure session established between the communicating endpoints. The session is identified by a GUID that lasts the lifetime of the security session. The following table shows what is kept and where.

DATA	STORAGE
Presentation credentials, such as username, X.509 certificates, Kerberos tokens, and references to credentials.	Standard Windows credential management mechanisms such as the Windows certificate store.
User membership information, such as usernames and passwords.	ASP.NET membership providers.
Identity information about the service used to authenticate the service to clients.	Endpoint address of the service.
Caller information.	Auditing logs.

Auditing

Auditing records the success and failure of authentication and authorization events. Auditing records contain the following data: service URI, action URI, and the caller's identification.

Auditing also records when the administrator modifies the configuration of message logging (turning it on or off), because message logging may log application-specific data in headers and bodies. For Windows XP, a record is logged in the application event log. For Windows Vista and Windows Server 2003, a record is logged in the security event log.

Transactions

The transactions feature provides transactional services to a WCF application.

Transaction headers used in transaction propagation may contain Transaction IDs or Enlistment IDs, which are GUIDs.

The Transactions feature uses the Microsoft Distributed Transaction Coordinator (MSDTC) Transaction Manager (a Windows component) to manage transaction state. By default, communications between Transactions Managers are encrypted. Transaction Managers may log endpoint references, Transaction IDs, and Enlistment IDs as part of their durable state. The lifetime of this state is determined by the lifetime of the Transaction Manager's log file. The MSDTC service owns and maintains this log.

The Transactions feature implements the WS-Coordination and WS-Atomic Transaction standards.

Reliable Sessions

Reliable sessions in WCF provide the transfer of messages when transport or intermediary failures occur. They provide an exactly-once transfer of messages even when the underlying transport disconnects (for example, a TCP connection on a wireless network) or loses a message (an HTTP proxy dropping an outgoing or incoming message). Reliable sessions also recover message reordering (as may happen in the case of multipath routing), preserving the order in which the messages were sent.

Reliable sessions are implemented using the WS-ReliableMessaging (WS-RM) protocol. They add WS-RM headers that contain session information, which is used to identify all messages associated with a particular reliable session. Each WS-RM session has an identifier, which is a GUID.

No personal information is retained on the end-user's machine.

Queued Channels

Queues store messages from a sending application on behalf of a receiving application and later forward these messages to the receiving application. They help ensure the transfer of messages from sending applications to receiving applications when, for example, the receiving application is transient. WCF provides support for queuing by using Microsoft Message Queuing (MSMQ) as a transport.

The queued channels feature does not add headers to a message. Instead it creates a Message Queuing message with appropriate Message Queuing message properties set, and invokes Message Queuing methods to put the message in the Message Queuing queue. Message Queuing is an optional component that ships with Windows.

No information is retained on the end-user's machine by the queued channels feature, because it uses Message Queuing as the queuing infrastructure.

COM+ Integration

This feature wraps existing COM and COM+ functionality to create services that are compatible with WCF services. This feature does not use specific headers and it does not retain data on the end-user's machine.

COM Service Moniker

This provides an unmanaged wrapper to a standard WCF client. This feature does not have specific headers on the wire nor does it persist data on the machine.

Peer Channel

A peer channel enables development of multiparty applications using WCF. Multiparty messaging occurs in the context of a mesh. Meshes are identified by a name that nodes can join. Each node in the peer channel creates a TCP listener at a user-specified port and establishes connections with other nodes in the mesh to ensure resiliency. To connect to other nodes in the mesh, nodes also exchange some data, including the listener address and the machine's IP addresses, with other nodes in the mesh. Messages sent around in the mesh can contain security information that pertains to the sender to prevent message spoofing and tampering.

No personal information is stored on the end-user's machine.

IT Professional Experience

Tracing

The diagnostics feature of the WCF infrastructure logs messages that pass through the transport and service model layers, and the activities and events associated with these messages. This feature is turned off by default. It is enabled using the application's configuration file and the tracing behavior may be modified using the WCF WMI provider at run time. When enabled, the tracing infrastructure emits a diagnostic trace that contains messages, activities, and processing events to configured listeners. The format and location of the output are determined by

the administrator's listener configuration choices, but is typically an XML formatted file. The administrator is responsible for setting the access control list (ACL) on the trace files. In particular, when hosted by Windows Activation System (WAS), the administrator should make sure the files are not served from the public virtual root directory if that is not desired.

There are two types of tracing: Message logging and Service Model diagnostic tracing, described in the following section. Each type is configured through its own trace source: [MessageLogging](#) and [System.ServiceModel](#). Both of these logging trace sources capture data that is local to the application.

Message Logging

The message logging trace source ([MessageLogging](#)) allows an administrator to log the messages that flow through the system. Through configuration, the user may decide to log entire messages or message headers only, whether to log at the transport and/or service model layers, and whether to include malformed messages. Also, the user may configure filtering to restrict which messages are logged.

By default, message logging is disabled. The local machine administrator can prevent the application-level administrator from turning message logging on.

Encrypted and Decrypted Message Logging

Messages are logged, encrypted, or decrypted, as described in the following terms.

Transport Logging

Logs messages received and sent at the transport level. These messages contain all headers, and may be encrypted before being sent on the wire and when being received.

If messages are encrypted before being sent on the wire and when they are received, they are logged encrypted as well. An exception is when a security protocol is used (https): they are then logged decrypted before being sent and after being received even if they are encrypted on the wire.

Service Logging

Logs messages received or sent at the service model level, after channel header processing has occurred, just before and after entering user code.

Messages logged at this level are decrypted even if they were secured and encrypted on the wire.

Malformed Message Logging

Logs messages that the WCF infrastructure cannot understand or process.

Messages are logged as-is, that is, encrypted or not

When messages are logged in decrypted or unencrypted form, by default WCF removes security keys and potentially personal information from the messages before logging them. The next sections describe what information is removed, and when. The machine administrator and application deployer must both take certain configuration actions to change the default behavior to log keys and potentially personal information.

Information Removed from Message Headers When Logging Decrypted/Unencrypted Messages

When messages are logged in decrypted/unencrypted form, security keys and potentially personal information are removed by default from message headers and message bodies before they are logged. The following list shows what WCF considers keys and potentially personal information.

Keys that are removed:

- For xmlns:wst="http://schemas.xmlsoap.org/ws/2004/04/trust" and xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust"

wst:BinarySecret

wst:Entropy

- For xmlns:wss="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.1.xsd" and

xmlns:wsse="http://docs.oasis-open.org/wss/2005/xx/oasis-2005xx-wss-wssecurity-secext-1.1.xsd"

wsse:Password

wsse:Nonce

Potentially personal information that is removed:

- For xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.1.xsd" and
xmlns:wssse="http://docs.oasis-open.org/wss/2005/xx/oasis-2005xx-wss-wssecurity-secext-1.1.xsd"

wsse:Username

wsse:BinarySecurityToken

- For xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion" the items in bold (below) are removed:

<Assertion

MajorVersion="1"

MinorVersion="1"

AssertionId="[ID]"

Issuer="[string]"

IssueInstant="[dateTime]"

<Conditions NotBefore="[dateTime]" NotOnOrAfter="[dateTime]">

<AudienceRestrictionCondition>

<Audience>[uri]</Audience> +

</AudienceRestrictionCondition>*

<DoNotCacheCondition />*

<!-- abstract base type

<Condition />*

-->

</Conditions>?

<Advice>

<AssertionIDReference>[ID]</AssertionIDReference>*

<Assertion>[assertion]</Assertion>*

[any]*

</Advice>?

<!-- Abstract base types

<Statement />*

<SubjectStatement>

<Subject>

<NameIdentifier

NameQualifier="[string]"?

Format="[uri]"?

>

[string]

</NameIdentifier>?

<SubjectConfirmation>

<ConfirmationMethod>[anyUri]</ConfirmationMethod> +

<SubjectConfirmationData>[any]</SubjectConfirmationData>?

<ds:KeyInfo>...</ds:KeyInfo>?

</SubjectConfirmation>?

</Subject>

</SubjectStatement>*

-->

<AuthenticationStatement

AuthenticationMethod="[uri]"

AuthenticationInstant="[dateTime]"

[Subject]

<SubjectLocality

IPAddress="[string]"?

DNSAddress="[string]"?

/>?

<AuthorityBinding

AuthorityKind="[QName]"

Location="[uri]"

Binding="[uri]"

/>*

</AuthenticationStatement>*

<AttributeStatement>

[Subject]

<Attribute

AttributeName="[string]"

AttributeNamespace="[uri]"

```
<AttributeValue>[any]</AttributeValue>+
```

```
</Attribute> +
```

```
</AttributeStatement> *
```

```
<AuthorizationDecisionStatement
```

```
Resource="[uri]"
```

```
Decision="[Permit|Deny|Indeterminate]"
```

```
[Subject]
```

```
<Action Namespace="[uri]">[string]</Action> +
```

```
<Evidence>
```

```
<AssertionIDReference>[ID]</AssertionIDReference> +
```

```
<Assertion>[assertion]</Assertion> +
```

```
</Evidence>?
```

```
</AuthorizationDecisionStatement> *
```

```
</Assertion>
```

Information Removed from Message Bodies When Logging Decrypted/Unencrypted Messages

As previously described, WCF removes keys and known potentially personal information from message headers for logged decrypted/unencrypted messages. In addition, WCF removes keys and known potentially personal information from message bodies for the body elements and actions in the following list, which describe security messages involved in key exchange.

For the following namespaces:

xmlns:wst="http://schemas.xmlsoap.org/ws/2004/04/trust" and

xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust" (for example, if no action available)

Information is removed for these body elements, which involve key exchange:

wst:RequestSecurityToken

wst:RequestSecurityTokenResponse

wst:RequestSecurityTokenResponseCollection

Information is also removed for each of the following Actions:

- `http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Issue`
- `http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Issue`
- `http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Renew`
- `http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Renew`
- `http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Cancel`
- `http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Cancel`
- `http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Validate`
- `http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Validate`

- `http://schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT`
- `http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/SCT`
- `http://schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT/Amend`
- `http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/SCT/Amend`
- `http://schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT/Renew`
- `http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/SCT/Renew`
- `http://schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT/Cancel`
- `http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/SCT/Cancel`
- `http://schemas.xmlsoap.org/ws/2004/04/security/trust/RST/SCT`
- `http://schemas.xmlsoap.org/ws/2004/04/security/trust/RSTR/SCT`
- `http://schemas.xmlsoap.org/ws/2004/04/security/trust/RST/SCT-Amend`
- `http://schemas.xmlsoap.org/ws/2004/04/security/trust/RSTR/SCT-Amend`

No Information Is Removed from Application-specific Headers and Body Data

WCF does not track personal information in application-specific headers (for example, query strings) or body data (for example, credit card number).

When message logging is on, personal information in application-specific headers and body information may be visible in the logs. Again, the application deployer is responsible for setting the ACLs on the configuration and log files. He also can turn off logging if he does not want this information to be visible, or he may filter out this information from the log files after it is logged.

Service Model Tracing

The Service Model trace source ([System.ServiceModel](#)) enables tracing of activities and events related to message processing. This feature uses the .NET Framework diagnostic functionality from [System.Diagnostics](#). As with the [MessageLogging](#) property, the location and its ACL are user-configurable using .NET Framework application configuration files. As with message logging, the file location is always configured when the administrator enables tracing; thus, the administrator controls the ACL.

Traces contain message headers when a message is in scope. The same rules for hiding potentially personal information in message headers in the previous section apply: the personal information previously identified is removed by default from the headers in traces. Both the machine administrator and the application deployer must modify the configuration in order to log potentially personal information. However, personal information contained in application-specific headers is logged in traces. The application deployer is responsible for setting the ACLs on the configuration and trace files. He also can turn off tracing if he does not want this information to be visible, or he can filter out this information from the trace files after it is logged.

As part of ServiceModel Tracing, Unique IDs (called Activity IDs, and typically a GUID) link different activities together as a message flows through different parts of the infrastructure.

Custom Trace Listeners

For both message logging and tracing, a custom trace listener can be configured, which can send traces and messages on the wire (for example, to a remote database). The application deployer is responsible for configuring custom listeners or enabling users to do so. He is also responsible for any personal information exposed at the remote location, and for properly applying ACLs to this location.

Other features for IT Professionals

WCF has a WMI provider that exposes the WCF infrastructure configuration information through WMI (shipped

with Windows). By default, the WMI interface is available to administrators.

WCF configuration uses the .NET Framework configuration mechanism. The configuration files are stored on the machine. The application developer and the administrator create the configuration files and ACL for each of the application's requirements. A configuration file can contain endpoint addresses and links to certificates in the certificate store. The certificates can be used to provide application data to configure various properties of the features used by the application.

WCF also uses the .NET Framework process dump functionality by calling the [FailFast](#) method.

IT Pro Tools

WCF also provides the following IT professional tools, which ship in the Windows SDK.

SvcTraceViewer.exe

The viewer displays WCF trace files. The viewer shows whatever information is contained in the traces.

SvcConfigEditor.exe

The editor allows the user to create and edit WCF configuration files. The editor shows whatever information is contained in the configuration files. The same task can be accomplished with a text editor.

ServiceModel_Reg

This tool allows the user to manage ServiceModel installs on a machine. The tool displays status messages in a console window when it runs and, in the process, may display information about the configuration of the WCF installation.

WSATConfig.exe and WSATUI.dll

These tools allow IT Professionals to configure interoperable WS-AtomicTransaction network support in WCF. The tools display and allow the user to change the values of the most commonly used WS-AtomicTransaction settings stored in the registry.

Cross-cutting Features

The following features are cross-cutting. That is, they can be composed with any of the preceding features.

Service Framework

Headers can contain an instance ID, which is a GUID that associates a message with an instance of a CLR class.

The Web Services Description Language (WSDL) contains a definition of the port. Each port has an endpoint address and a binding that represents the services used by the application. Exposing WSDL can be turned off using configuration. No information is retained on the machine.

See Also

[Windows Communication Foundation Security](#)