**ASP.NET Impersonation**

**.NET Framework 4**

[Other Versions](javascript:;)

http://i.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/xh507fc5(d=printer,v=vs.90).ASPX)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/xh507fc5(d=printer,v=vs.85).ASPX)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/xh507fc5(d=printer,v=vs.80).ASPX)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/xh507fc5(d=printer,v=vs.71).ASPX)

When using impersonation, ASP.NET applications can execute with the Windows identity (user account) of the user making the request. Impersonation is commonly used in applications that rely on Microsoft Internet Information Services (IIS) to authenticate the user.

ASP.NET impersonation is disabled by default. If impersonation is enabled for an ASP.NET application, that application runs in the context of the identity whose access token IIS passes to ASP.NET. That token can be either an authenticated user token, such as a token for a logged-in Windows user, or the token that IIS provides for anonymous users (typically, the IUSR\_MACHINENAME identity).

When impersonation is enabled, only your application code runs under the context of the impersonated user. Applications are compiled and configuration information is loaded using the identity of the ASP.NET process. For more information, see [Configuring ASP.NET Process Identity](http://msdn.microsoft.com/en-us/library/dwc1xthy(v=vs.100).ASPX). The compiled application is put in the Temporary ASP.NET files directory. The application identity that is being impersonated needs to have read/write access to this directory. The impersonated application identity also requires at least read access to the files in your application directory and subdirectories. For more information, see [ASP.NET Required Access Control Lists (ACLs)](http://msdn.microsoft.com/en-us/library/kwzs111e(v=vs.100).ASPX).

|  |
| --- |
| **NoteNote** |
| Because ASP.NET uses the Windows identity of the ASP.NET process when compiling applications and loading configuration information, you must keep application code and configuration information private between applications on a server that hosts multiple applications. On Windows Server 2003 you can create multiple application pools and specify a unique identity for each application pool. You can then restrict access to application files using access control lists (ACLs) (if file system is formatted using NTFS) and these identities. For example, consider two applications, App1 and App2, where the information in each application must be kept private. You can put App1 in the ApplicationPool1 application pool which has an identity of ID\_ApplicationPool1. You can put App2 in the ApplicationPool2 application pool which has an identity of ID\_ApplicationPool2. The ID\_ApplicationPool1 account is given access to the files in App1, but denied access to the files in App2. ID\_ApplicationPool2 is given access to the files in App2, but denied access to the files in App1. Note that you cannot make this separation on Windows 2000 or Windows XP Professional, because on those operating systems, the process identity for all ASP.NET applications is a single identity. |

You control impersonation using the [identity](http://msdn.microsoft.com/en-us/library/72wdk8cc(v=vs.100).ASPX) configuration element. As with other configuration directives, this directive applies hierarchically. A minimal configuration file to enable impersonation for an application might look like the following example:

<configuration>

<system.web>

<identity impersonate="true"/>

</system.web>

</configuration>

You can also add support for specific names to run an application as a configurable identity, as shown in the following example:

<identity impersonate="true"

userName="contoso\Jane"

password="\*\*\*\*\*\*\*\*" />

Substitute the correct password for the value listed in the previous example.

|  |
| --- |
| **NoteNote** |
| In the preceding example, the user name and password are stored in clear text in the configuration file. To improve the security of your application, it is recommended that you restrict the access to your Web.config file using an Access Control List (ACL) and that you encrypt the [identity](http://msdn.microsoft.com/en-us/library/72wdk8cc(v=vs.100).ASPX)configuration element in your Web.config file using protected configuration. For more information, see [Encrypting Configuration Information Using Protected Configuration](http://msdn.microsoft.com/en-us/library/53tyfkaw(v=vs.100).ASPX). |

The configuration illustrated in the example enables the entire application to run using the contoso\Jane identity, regardless of the identity of the request. This type of impersonation can be delegated to another computer. That is, if you specify the user name and password for the impersonated user, you can connect to another computer on the network and request resources, such as files or access to SQL Server, using integrated security. If you enable impersonation and do not specify a domain account as the identity, you will not be able to connect to another computer on the network unless your IIS application is configured to use Basic authentication.

|  |
| --- |
| **NoteNote** |
| On Windows 2000, you cannot impersonate using specific user credentials for the identity of the ASP.NET worker process. But you can enable impersonation without specific user credentials so that your application impersonates the identity determined by IIS. For more information, see article 810204, "PRB: Per Request Impersonation Does Not Work on Windows 2000 with ASP.NET," in the Microsoft Knowledge Base at http://support.microsoft.com. |

[Reading the Impersonated Identity](javascript:void(0))

The following code example shows how to programmatically read the identity of the impersonated user:

C#

[VB](http://msdn.microsoft.com/en-us/library/xh507fc5(d=printer,v=vs.100).ASPX?cs-save-lang=1&cs-lang=vb#code-snippet-3)

String username =

System.Security.Principal.WindowsIdentity.GetCurrent().Name;

**Impersonating and Reverting**

**.NET Framework 4.5**

[Other Versions](javascript:;)



* [.NET Framework 4](http://msdn.microsoft.com/en-us/library/b80a7e92(d=printer,v=vs.100).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/b80a7e92(d=printer,v=vs.85).aspx)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/b80a7e92(d=printer,v=vs.71).aspx)
* [.NET Framework 3.5](http://msdn.microsoft.com/en-us/library/b80a7e92(d=printer,v=vs.90).aspx)
* [.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/b80a7e92(d=printer,v=vs.80).aspx)

Sometimes you might need to obtain a Windows NT account token to impersonate a Windows account. For example, your ASP.NET-based application might have to act on behalf of several users at different times. Your application might accept a token that represents an administrator from Internet Information Services (IIS), impersonate that user, perform an operation, and revert to the previous identity. Next, it might accept a token from IIS that represents a user with fewer rights, perform some operation, and revert again.

In situations where your application must impersonate a Windows account that has not been attached to the current thread by IIS, you must retrieve that account's token and use it to activate the account. You can do this by performing the following tasks:

1. Retrieve an account token for a particular user by making a call to the unmanaged **LogonUser** method. This method is not in the .NET Framework base class library, but is located in the unmanaged **advapi32.dll**. Accessing methods in unmanaged code is an advanced operation and is beyond the scope of this discussion. For more information, see [Interoperating with Unmanaged Code](http://msdn.microsoft.com/en-us/library/sd10k43k(v=vs.110).aspx). For more information about the **LogonUser** method and **advapi32.dll**, see the Platform SDK documentation.
2. Create a new instance of the **WindowsIdentity** class, passing the token. The following code demonstrates this call, where hToken represents a Windows token.

C#

[VB](http://msdn.microsoft.com/en-us/library/b80a7e92(d=printer,v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

WindowsIdentity ImpersonatedIdentity = new WindowsIdentity(hToken);

1. Begin impersonation by creating a new instance of the [WindowsImpersonationContext](http://msdn.microsoft.com/en-us/library/system.security.principal.windowsimpersonationcontext(v=vs.110).aspx) class and initializing it with the [WindowsIdentity.Impersonate](http://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity.impersonate(v=vs.110).aspx) method of the initialized class, as shown in the following code.

C#

[VB](http://msdn.microsoft.com/en-us/library/b80a7e92(d=printer,v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

WindowsImpersonationContext MyImpersonation = ImpersonatedIdentity.Impersonate();

1. When you no longer need to impersonate, call the [WindowsImpersonationContext.Undo](http://msdn.microsoft.com/en-us/library/system.security.principal.windowsimpersonationcontext.undo(v=vs.110).aspx) method to revert the impersonation, as shown in the following code.

C#

[VB](http://msdn.microsoft.com/en-us/library/b80a7e92(d=printer,v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

MyImpersonation.Undo();

If trusted code has already attached a [WindowsPrincipal](http://msdn.microsoft.com/en-us/library/system.security.principal.windowsprincipal(v=vs.110).aspx) object to the thread, you can call the instance method **Impersonate**, which does not take an account token. Note that this is only useful when the **WindowsPrincipal** object on the thread represents a user other than the one under which the process is currently executing. For example, you might encounter this situation using ASP.NET with Windows authentication turned on and impersonation turned off. In this case, the process is running under an account configured in Internet Information Services (IIS) while the current principal represents the Windows user that is accessing the page.

Note that neither **Impersonate** nor **Undo** changes the [Principal](http://msdn.microsoft.com/en-us/library/system.security.principal.iprincipal(v=vs.110).aspx) object associated with the current call context. Rather, impersonation and reverting change the token associated with the current operating system process..

**Using IIS Authentication with ASP.NET Impersonation**

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/134ec8tc(d=printer,v=vs.90).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/134ec8tc(d=printer,v=vs.85).aspx)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/134ec8tc(d=printer,v=vs.80).aspx)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/134ec8tc(d=printer,v=vs.71).aspx)

Internet Information Services (IIS) provides several authentication schemes that can be employed when securing a Web application. Common scenarios include using Integrated Windows authentication (NTLM) within a corporate intranet to determine application users' identity based on their Windows login, or specifying a single anonymous identity for a particular application. The Windows identity supplied by IIS can then be used to determine whether the Web application has access to a protected Windows resource, such as a file protected using an Access Control List (ACL), or a network resource such as a file or database server. You can configure ASP.NET to use the Windows identity supplied by IIS using impersonation.

By default, ASP.NET is configured to use Windows authentication mode, which applies the Windows identity supplied by IIS to the [User](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.user(v=vs.100).aspx) property of the current [HttpContext](http://msdn.microsoft.com/en-us/library/system.web.httpcontext(v=vs.100).aspx) object. This enables you to determine the identity supplied by IIS through the [User](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.user(v=vs.100).aspx) property (the user [Name](http://msdn.microsoft.com/en-us/library/system.security.principal.iidentity.name(v=vs.100).aspx) is blank when anonymous identification is used), but does not use the supplied identity as the [WindowsIdentity](http://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity(v=vs.100).aspx) for the current page. The [WindowsIdentity](http://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity(v=vs.100).aspx) for an application is used when determining if the application has access to a particular file or network resource.

To configure ASP.NET to impersonate the Windows identity supplied by IIS as the [WindowsIdentity](http://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity(v=vs.100).aspx) for the ASP.NET application, edit the Web.config file for the application and set the impersonate attribute of the identity configuration element to true, as shown in the following example.

<configuration>

<system.web>

<identity impersonate="true" />

</system.web>

</configuration>

Impersonation is independent of the authentication mode configured using the [authentication](http://msdn.microsoft.com/en-us/library/532aee0e(v=vs.100).aspx) configuration element. The authentication element is used to determine the [User](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.user(v=vs.100).aspx) property of the current [HttpContext](http://msdn.microsoft.com/en-us/library/system.web.httpcontext(v=vs.100).aspx). Impersonation is used to determine the [WindowsIdentity](http://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity(v=vs.100).aspx) of the ASP.NET application.

The following describes how you would enable impersonation using an intranet scenario as an example. In this scenario, you are setting up an internal corporate Web site for posting employee information. However, some of the information is for managers only. The manager information can be posted to a subdirectory of the employee information site, so that access to the information can be limited. IIS determines the user's identity using Windows Integrated (NTLM) security. The scenario assumes that:

* The Web server has the Microsoft Windows NT Server, Windows 2000 Server, or Windows Server 2003 operating system installed.
* IIS 6.0 is installed the Web server.
* The Web server hard disk is formatted using NTFS.
* All employees that need access to restricted resources are using Windows.

As the administrator of the application in the scenario, you would need to do the following:

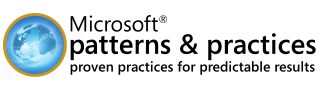
1. Create the files and directories shown in the following illustration:

134ec8tc.directories(en-us,VS.100).gif

1. Create a Windows group named Managers that contains all users who should have access to the ManagerInfo.aspx file.
2. Use Internet Information Services (IIS) Manager to disable anonymous authentication for the application and enable integrated windows authentication.
3. In the application's Web.config file, set the impersonate attribute in the [identity](http://msdn.microsoft.com/en-us/library/72wdk8cc(v=vs.100).aspx) element to true.
4. Set the NTFS access control list (ACL) for the ManagerInformation directory to allow access to only those identities that are in the Windows Manager group and any required system accounts. You would need to be sure to include the identity of the ASP.NET process. The identity of the ASP.NET process for Windows 2000 Server or Windows NT is the local ASPNET account. The identity of the ASP.NET process for Windows Server 2003 and later is the identity of the IIS application pool, which by default is the NETWORK SERVICE account.

|  |
| --- |
| **NoteNote** |
| The ASP.NET role-management feature provides an alternative method of restricting access to areas of your Web application. For more information, see [Managing Authorization Using Roles](http://msdn.microsoft.com/en-us/library/9ab2fxh0(v=vs.100).aspx). |

# Impersonation/Delegation

[](http://msdn.microsoft.com/en-us/practices/default.aspx)

* [What are my impersonation options?](http://msdn.microsoft.com/en-us/library/ff647248(d=printer).aspx#ImpersonationDelegation1)
* [What is the difference between impersonation and delegation?](http://msdn.microsoft.com/en-us/library/ff647248(d=printer).aspx#ImpersonationDelegation2)
* [How do I impersonate the original caller for an operation call?](http://msdn.microsoft.com/en-us/library/ff647248(d=printer).aspx#ImpersonationDelegation3)
* [How do I temporarily impersonate the original caller in an operation call?](http://msdn.microsoft.com/en-us/library/ff647248(d=printer).aspx#ImpersonationDelegation4)
* [How do I impersonate a specific (fixed) identity?](http://msdn.microsoft.com/en-us/library/ff647248(d=printer).aspx#ImpersonationDelegation5)
* [What is constrained delegation?](http://msdn.microsoft.com/en-us/library/ff647248(d=printer).aspx#ImpersonationDelegation6)
* [What is protocol transition?](http://msdn.microsoft.com/en-us/library/ff647248(d=printer).aspx#ImpersonationDelegation7)
* [How do I flow the original caller from the ASP.NET client to a WCF service?](http://msdn.microsoft.com/en-us/library/ff647248(d=printer).aspx#ImpersonationDelegation8)
* [What is the difference between declarative and programmatic impersonation?](http://msdn.microsoft.com/en-us/library/ff647248(d=printer).aspx#ImpersonationDelegation9)
* [What is the trusted subsystem model?](http://msdn.microsoft.com/en-us/library/ff647248(d=printer).aspx#ImpersonationDelegation10)
* [When should I flow the original caller to back-end code?](http://msdn.microsoft.com/en-us/library/ff647248(d=printer).aspx#ImpersonationDelegation11)
* [How do I control access to a remote resource based on the original caller's identity?](http://msdn.microsoft.com/en-us/library/ff647248(d=printer).aspx#ImpersonationDelegation12)

## What are my impersonation options?

There are three options for impersonation:

* **Impersonating the original caller declaratively on specific operations**. Use this option when you want to impersonate the original caller for the entire duration of a specific operation.
* **Impersonating the original caller declaratively on the entire service**. Use this option when you want to impersonate the original caller for the entire duration of all operations in the service.
* **Impersonating the original caller programmatically within an operation**. Use this option when you want to impersonate the original caller for a short duration in a service operation.

### Additional Resources

* For more information, see [Delegation and Impersonation with WCF](http://msdn.microsoft.com/en-us/library/ms730088.aspx).

## What is the difference between impersonation and delegation?

Impersonation flows the original caller's identity to back-end resources on the same computer. Delegation flows the original caller's identity to back-end resources on computers other than the computer running the service.

For example, if a service is running within IIS without impersonation, the service will access resources using the ASP.NET account in IIS 5.0, or the Network Service account in IIS 6.0. With impersonation, if the client is connecting using the original caller's account, the service will access resources such as a SQL Server database on the same machine using the original caller's account instead of the system ASP.NET account. Delegation is similar except that the SQL Server database could be on a different machine that is remote to the service.

### Additional Resources

* For more information, see [Delegation and Impersonation with WCF](http://msdn.microsoft.com/en-us/library/ms730088.aspx).

## How do I impersonate the original caller for an operation call?

Because impersonation is a costly operation and is usually used for higher-privileged original callers, you should use impersonation only on operations that need it to reduce the potential attack surface.

You can impersonate declaratively by applying the **OperationBehaviorAttribute** attribute on any operation that requires client impersonation, as shown in the following code example:

[OperationBehavior(Impersonation = ImpersonationOption.Required)]

public string GetData(int value)

{

return “test”;

}

### Additional Resources

* For more information, see [Delegation and Impersonation with WCF](http://msdn.microsoft.com/en-us/library/ms730088.aspx).

## How do I temporarily impersonate the original caller in an operation call?

Because impersonation is a costly operation and is usually used for higher-privileged original callers, you should use impersonation only when it is needed to reduce the potential attack surface. Programmatic impersonation allows you to impersonate on specific lines of code rather than the entire operation.

You can use programmatic impersonation to temporarily impersonate the original caller in an operation call, as shown in the following example:

public string GetData(int value)

{

using (ServiceSecurityContext.Current.WindowsIdentity.Impersonate())

{

// return the impersonated user (original users identity)

return string.Format("Hi, {0}, you have entered: {1}",

WindowsIdentity.GetCurrent().Name, value);

}

}

In the above example, the **using** statement is employed to ensure that the impersonation is reverted after execution of the **using** block. It is important to revert impersonation because failure to do so can form the basis for denial of service (DoS) and elevation of privilege attacks.

### Additional Resources

* For more information, see [Delegation and Impersonation with WCF](http://msdn.microsoft.com/en-us/library/ms730088.aspx).

## How do I impersonate a specific (fixed) identity?

Use the **WindowsIdentity** class to obtain a Windows token and logon session for a given domain account by supplying a user principal name (UPN). With this approach, you do not need the account's password.

using System.Security.Principal;

…

WindowsIdentity wi = new WindowsIdentity(userName@fullyqualifieddomainName);

WindowsImpersonationContext ctx = null;

try

{

ctx = wi.Impersonate();

// Thread is now impersonating you can call the backend operations here...

catch

{

// Prevent exceptions propagating.

}

finally

{

// Ensure impersonation is reverted

ctx.Undo();

}

|  |
| --- |
| **Ff647248.note(en-us,PandP.10).gifNote:** |
| The **WindowsIdentity** constructor relies on a Windows Server 2003 extension to the Kerberos protocol called Service for User to Self (S4U2Self). You can use this approach if your application runs on a Windows Server 2003 server in a Windows Server 2003 domain. The advantage of this approach is that you do not have to store credentials as you do for LogonUser. |

### Additional Resources

* For more information, see [Delegation and Impersonation with WCF](http://msdn.microsoft.com/en-us/library/ms730088.aspx).

## What is constrained delegation?

Impersonation is a WCF service configuration in which the service will access resources on the same computer using a client's user identity. Delegation is similar to impersonation except that the WCF service can access resources that are on the same machine or on other machines using the client's user identity. Delegation flows the original caller's identity to back-end resources on the computers other than the computer running the service.

The Microsoft Windows Server 2003 operating system provides a more secure form of delegation called constrained delegation. With constrained delegation, you can configure the Microsoft Active Directory directory service to restrict the services and servers that your WCF service application can access with the impersonated identity. Constrained delegation in Windows Server 2003 requires Kerberos authentication.

### Additional Resources

* For more information, see [Delegation and Impersonation with WCF](http://msdn.microsoft.com/en-us/library/ms730088.aspx).

## What is protocol transition?

Protocol transition is a Windows Server 2003 feature that allows you to switch from an alternate, non-Windows authentication mode (such as Forms-based or certificate authentication) to Kerberos authentication. This is useful when your application cannot use Kerberos authentication to authenticate its callers, and when your application needs to use constrained delegation to access downstream network resources.

### Additional Resources

* For more information, see [Delegation and Impersonation with WCF](http://msdn.microsoft.com/en-us/library/ms730088.aspx).

## How do I flow the original caller from the ASP.NET client to a WCF service?

The following steps show how to impersonate the original caller from the ASP.NET client to a WCF service:

1. Configure your WCF service to use Windows authentication.

…

<services>

<service name="Service" behaviorConfiguration="ServiceBehavior">

<endpoint address="" binding="wsHttpBinding" contract="IService">

<identity>

<dns value="localhost"/>

</identity>

</endpoint>

<endpoint address="mex" binding="mexHttpBinding"

contract="IMetadataExchange"/>

</service>

</services>

…

1. Configure the SPN identity for the WCF service endpoint.

<endpoint address="" binding="wsHttpBinding" contract="IService">

<identity>

<servicePrincipalName value="HOST/YourMachineName" />

<dns value="" />

</identity>

</endpoint>

1. Implement impersonation in the WCF service.

using System.Security.Principal;

[OperationBehavior(Impersonation = ImpersonationOption.Required)]

public string GetData(int value)

{

return string.Format("Hi, {0}, you have entered: {1}",

WindowsIdentity.GetCurrent().Name, value);

}

1. Create a Web application test client and add the WCF service reference.
2. Impersonate the original caller when calling the WCF service.

using System.Security.Principal;

…

protected void Button1\_Click(object sender, EventArgs e)

{

// Obtain the authenticated user's Identity and impersonate the original caller

using (((WindowsIdentity)HttpContext.Current.User.Identity).Impersonate())

{

WCFTestService.ServiceClient myService = new WCFTestService.ServiceClient();

Response.Write(myService.GetData(123) + "<br/>");

myService.Close();

}

}

…

1. Configure the Web application for constrained delegation.
   * If your ASP.NET application runs using the Network Service machine account, you must enable constrained delegation for your Web server computer.
   * If your ASP.NET application runs under a custom domain account, you must enable protocol transition and constrained delegation for the custom domain account.
2. Test the client and WCF service.

### Additional Resources

* For more information on constrained delegation, see [How To: Use Protocol Transition and Constrained Delegation in ASP.NET 2.0](http://msdn.microsoft.com/en-us/library/ms998355.aspx).

## What is the difference between declarative and programmatic impersonation?

Impersonation is used to restrict or authorize the original caller's access to a WCF service's local resources, such as files. Use declarative impersonation to define impersonation at the operation or service level.

Impersonate declaratively by applying the **OperationBehaviorAttribute** attribute on any operation that requires client impersonation, as shown in the following code example:

[OperationBehavior(Impersonation = ImpersonationOption.Required)]

public string GetData(int value)

{

return “test”;

}

Use programmatic impersonation to define finer-grained impersonation based on business logic. Programmatic impersonation is specified in code and applied at run time.

Programmatic impersonation can be performed as shown in the following example:

public string GetData(int value)

{

using (ServiceSecurityContext.Current.WindowsIdentity.Impersonate())

{

// return the impersonated user (original users identity)

return string.Format("Hi, {0}, you have entered: {1}",

WindowsIdentity.GetCurrent().Name, value);

}

}

### Additional Resources

* For more information, see [Delegation and Impersonation with WCF](http://msdn.microsoft.com/en-us/library/ms730088.aspx).

## What is the trusted subsystem model?

A trusted subsystem describes an architecture in which an upstream tier is trusted to authenticate and authorize the original caller for downstream components. For instance, a database server trusts the Web application to authenticate users, and then all calls from the Web application to the database server are made with the Web application's identity instead of the original caller's identity. In this model, the web application's identity is trusted to make calls on behalf of the original caller.

The advantages of the trusted subsystem model include support for efficient connection pooling, no direct data access because only the service account is granted access to the back-end resources, and minimal back-end access control list (ACL) management.

### Additional Resources

* For more information, see [Trusted Subsystem](http://msdn.microsoft.com/en-us/library/ms730288.aspx).

## When should I flow the original caller to back-end code?

Flow the original caller to back-end code when you need to authorize access to resources based on the original caller's identity, or when the back-end code needs to perform roles-based authorization.

### Additional Resources

* For more information, see [Delegation and Impersonation with WCF](http://msdn.microsoft.com/en-us/library/ms730088.aspx).

## How do I control access to a remote resource based on the original caller's identity?

Use delegation to flow the impersonated original user's security context (Windows identity) to the remote back-end service. On the remote back-end service, the original user's Windows identity can be used to authenticate or impersonate the original caller in order to restrict or authorize the original caller's access to local resources.

When using delegation on Windows Server 2003 or later, use constrained delegation. This allows administrators to specify exactly which services can be accessed on a downstream server or a domain account when using an impersonated user's security context.

### Additional Resources

* For more information, see [Delegation and Impersonation with WCF](http://msdn.microsoft.com/en-us/library/ms730088.aspx).

# Delegation and Impersonation with WCF

**.NET Framework 4.5**

[Other Versions](javascript:;)



* [.NET Framework 4](http://msdn.microsoft.com/en-us/library/ms730088(d=printer,v=vs.100).aspx)
* [.NET Framework 3.5](http://msdn.microsoft.com/en-us/library/ms730088(d=printer,v=vs.90).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/ms730088(d=printer,v=vs.85).aspx)

Impersonation is a common technique that services use to restrict client access to a service domain's resources. Service domain resources can either be machine resources, such as local files (impersonation), or a resource on another machine, such as a file share (delegation). For a sample application, see [Impersonating the Client](http://msdn.microsoft.com/en-us/library/ms751513.aspx). For an example of how to use impersonation, see [How to: Impersonate a Client on a Service](http://msdn.microsoft.com/en-us/library/ms731090.aspx).

|  |
| --- |
| **ImportantNote:** |
| Be aware that when impersonating a client on a service, the service runs with the client's credentials, which may have higher privileges than the server process. |

## Overview

Typically, clients call a service to have the service perform some action on the client’s behalf. Impersonation allows the service to act as the client while performing the action. Delegation allows a front-end service to forward the client’s request to a back-end service in such a way that the back-end service can also impersonate the client. Impersonation is most commonly used as a way of checking whether a client is authorized to perform a particular action, while delegation is a way of flowing impersonation capabilities, along with the client’s identity, to a back-end service. Delegation is a Windows domain feature that can be used when Kerberos-based authentication is performed. Delegation is distinct from identity flow and, because delegation transfers the ability to impersonate the client without possession of the client’s password, it is a much higher privileged operation than identity flow.

Both impersonation and delegation require that the client have a Windows identity. If a client does not possess a Windows identity, then the only option available is to flow the client’s identity to the second service.

## Impersonation Basics

Windows Communication Foundation (WCF) supports impersonation for a variety of client credentials. This topic describes service model support for impersonating the caller during the implementation of a service method. Also discussed are common deployment scenarios involving impersonation and SOAP security and WCF options in these scenarios.

This topic focuses on impersonation and delegation in WCF when using SOAP security. You can also use impersonation and delegation with WCF when using transport security, as described in [Using Impersonation with Transport Security](http://msdn.microsoft.com/en-us/library/ms788971.aspx).

## Two Methods

WCF SOAP security has two distinct methods for performing impersonation. The method used depends on the binding. One is impersonation from a Windows token obtained from the Security Support Provider Interface (SSPI) or Kerberos authentication, which is then cached on the service. The second is impersonation from a Windows token obtained from the Kerberos extensions, collectively called Service-for-User (S4U).

### Cached Token Impersonation

You can perform cached-token impersonation with the following:

* [WSHttpBinding](http://msdn.microsoft.com/en-us/library/system.servicemodel.wshttpbinding.aspx), [WSDualHttpBinding](http://msdn.microsoft.com/en-us/library/system.servicemodel.wsdualhttpbinding.aspx), and [NetTcpBinding](http://msdn.microsoft.com/en-us/library/system.servicemodel.nettcpbinding.aspx) with a Windows client credential.
* [BasicHttpBinding](http://msdn.microsoft.com/en-us/library/system.servicemodel.basichttpbinding.aspx) with a [BasicHttpSecurityMode](http://msdn.microsoft.com/en-us/library/system.servicemodel.basichttpsecuritymode.aspx) set to the [TransportWithMessageCredential](http://msdn.microsoft.com/en-us/library/system.servicemodel.basichttpsecuritymode.aspx) credential, or any other standard binding where the client presents a user name credential that the service can map to a valid Windows account.
* Any [CustomBinding](http://msdn.microsoft.com/en-us/library/system.servicemodel.channels.custombinding.aspx) that uses a Windows client credential with the **requireCancellation** set to **true**. (The property is available on the following classes: [SecureConversationSecurityTokenParameters](http://msdn.microsoft.com/en-us/library/system.servicemodel.security.tokens.secureconversationsecuritytokenparameters.aspx), [SslSecurityTokenParameters](http://msdn.microsoft.com/en-us/library/system.servicemodel.security.tokens.sslsecuritytokenparameters.aspx), and [SspiSecurityTokenParameters](http://msdn.microsoft.com/en-us/library/system.servicemodel.security.tokens.sspisecuritytokenparameters.aspx).) If a secure conversation is used on the binding, it must also have the **requireCancellation** property set to **true**.
* Any [CustomBinding](http://msdn.microsoft.com/en-us/library/system.servicemodel.channels.custombinding.aspx) where the client presents a user name credential. If secure conversation is used on the binding, it must also have the **requireCancellation** property set to **true**.

### S4U-Based Impersonation

You can perform S4U-based impersonation with the following:

* [WSHttpBinding](http://msdn.microsoft.com/en-us/library/system.servicemodel.wshttpbinding.aspx), [WSDualHttpBinding](http://msdn.microsoft.com/en-us/library/system.servicemodel.wsdualhttpbinding.aspx), and [NetTcpBinding](http://msdn.microsoft.com/en-us/library/system.servicemodel.nettcpbinding.aspx) with a certificate client credential that the service can map to a valid Windows account.
* Any [CustomBinding](http://msdn.microsoft.com/en-us/library/system.servicemodel.channels.custombinding.aspx) that uses a Windows client credential with the **requireCancellation** property set to **false**.
* Any [CustomBinding](http://msdn.microsoft.com/en-us/library/system.servicemodel.channels.custombinding.aspx) that uses a user name or Windows client credential and secure conversation with the **requireCancellation** property set to **false**.

The extent to which the service can impersonate the client depends on the privileges the service account holds when it attempts impersonation, the type of impersonation used, and possibly the extent of impersonation the client permits.

|  |
| --- |
| **noteNote:** |
| When the client and service are running on the same computer and the client is running under a system account (for example, **Local System** or **Network Service**), the client cannot be impersonated when a secure session is established with stateful Security Context tokens. A Windows Form or console application typically runs 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 (SCT) by default. However, if the client is an ASP.NET page, and secure sessions with stateful SCTs are used, the client cannot be impersonated. For more information about using stateful SCTs in a secure session, see [How to: Create a Security Context Token for a Secure Session](http://msdn.microsoft.com/en-us/library/ms731814.aspx). |

## Impersonation in a Service Method: Declarative Model

Most impersonation scenarios involve executing the service method in the caller context. WCF provides an impersonation feature that makes this easy to do by allowing the user to specify the impersonation requirement in the [OperationBehaviorAttribute](http://msdn.microsoft.com/en-us/library/system.servicemodel.operationbehaviorattribute.aspx) attribute. For example, in the following code, the WCF infrastructure impersonates the caller before executing the Hello method. Any attempt to access native resources inside the Hello method succeed only if the access control list (ACL) of the resource allows the caller access privileges. To enable impersonation, set the [Impersonation](http://msdn.microsoft.com/en-us/library/system.servicemodel.operationbehaviorattribute.impersonation.aspx) property to one of the [ImpersonationOption](http://msdn.microsoft.com/en-us/library/system.servicemodel.impersonationoption.aspx) enumeration values, either [System.ServiceModel.ImpersonationOption.Required](http://msdn.microsoft.com/en-us/library/system.servicemodel.impersonationoption.aspx) or [System.ServiceModel.ImpersonationOption.Allowed](http://msdn.microsoft.com/en-us/library/system.servicemodel.impersonationoption.aspx), as shown in the following example.

|  |
| --- |
| **noteNote:** |
| When a service has higher credentials than the remote client, the credentials of the service are used if the [Impersonation](http://msdn.microsoft.com/en-us/library/system.servicemodel.operationbehaviorattribute.impersonation.aspx) property is set to [Allowed](http://msdn.microsoft.com/en-us/library/system.servicemodel.impersonationoption.aspx). That is, if a low-privileged user provides its credentials, a higher-privileged service executes the method with the credentials of the service, and can use resources that the low-privileged user would otherwise not be able to use. |

C#

[VB](http://msdn.microsoft.com/en-us/library/ms730088(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

[ServiceContract]

public interface IHelloContract

{

[OperationContract]

string Hello(string message);

}

public class HelloService : IHelloService

{

[OperationBehavior(Impersonation = ImpersonationOption.Required)]

public string Hello(string message)

{

return "hello";

}

}

The WCF infrastructure can impersonate the caller only if the caller is authenticated with credentials that can be mapped to a Windows user account. If the service is configured to authenticate using a credential that cannot be mapped to a Windows account, the service method is not executed.

|  |
| --- |
| **noteNote:** |
| On Windows XP, impersonation fails if a stateful SCT is created, resulting in an [InvalidOperationException](http://msdn.microsoft.com/en-us/library/system.invalidoperationexception.aspx). For more information, see [Unsupported Scenarios](http://msdn.microsoft.com/en-us/library/aa738624.aspx). |

## Impersonation in a Service Method: Imperative Model

Sometimes a caller does not need to impersonate the entire service method to function, but for only a portion of it. In this case, obtain the Windows identity of the caller inside the service method and imperatively perform the impersonation. Do this by using the [WindowsIdentity](http://msdn.microsoft.com/en-us/library/system.servicemodel.servicesecuritycontext.windowsidentity.aspx) property of the [ServiceSecurityContext](http://msdn.microsoft.com/en-us/library/system.servicemodel.servicesecuritycontext.aspx) to return an instance of the [WindowsIdentity](http://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity.aspx) class and calling the [Impersonate](http://msdn.microsoft.com/en-us/library/w070t6ka.aspx) method before using the instance.

|  |
| --- |
| **noteNote:** |
| Be sure to use the Visual Basic **Using** statement or the C# **using** statement to automatically revert the impersonation action. If you do not use the statement, or if you use a programming language other than Visual Basic or C#, be sure to revert the impersonation level. Failure to do this can form the basis for denial of service and elevation of privilege attacks. |

C#

[VB](http://msdn.microsoft.com/en-us/library/ms730088(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

public class HelloService : IHelloService

{

[OperationBehavior]

public string Hello(string message)

{

WindowsIdentity callerWindowsIdentity =

ServiceSecurityContext.Current.WindowsIdentity;

if (callerWindowsIdentity == null)

{

throw new InvalidOperationException

("The caller cannot be mapped to a WindowsIdentity");

}

using (callerWindowsIdentity.Impersonate())

{

// Access a file as the caller.

}

return "Hello";

}

}

## Impersonation for All Service Methods

In some cases, you must perform all the methods of a service in the caller’s context. Instead of explicitly enabling this feature on a per-method basis, use the [ServiceAuthorizationBehavior](http://msdn.microsoft.com/en-us/library/system.servicemodel.description.serviceauthorizationbehavior.aspx). As shown in the following code, set the [ImpersonateCallerForAllOperations](http://msdn.microsoft.com/en-us/library/system.servicemodel.description.serviceauthorizationbehavior.impersonatecallerforalloperations.aspx) property to **true**. The [ServiceAuthorizationBehavior](http://msdn.microsoft.com/en-us/library/system.servicemodel.description.serviceauthorizationbehavior.aspx) is retrieved from the collections of behaviors of the [ServiceHost](http://msdn.microsoft.com/en-us/library/system.servicemodel.servicehost.aspx) class. Also note that the **Impersonation** property of the [OperationBehaviorAttribute](http://msdn.microsoft.com/en-us/library/system.servicemodel.operationbehaviorattribute.aspx) applied to each method must also be set to either [Allowed](http://msdn.microsoft.com/en-us/library/system.servicemodel.impersonationoption.aspx) or [Required](http://msdn.microsoft.com/en-us/library/system.servicemodel.impersonationoption.aspx).

C#

[VB](http://msdn.microsoft.com/en-us/library/ms730088(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

// Code to create a ServiceHost not shown.

ServiceAuthorizationBehavior MyServiceAuthoriationBehavior =

serviceHost.Description.Behaviors.Find<ServiceAuthorizationBehavior>();

MyServiceAuthoriationBehavior.ImpersonateCallerForAllOperations = true;

The following table describes WCF behavior for all possible combinations of **ImpersonationOption** and **ImpersonateCallerForAllServiceOperations**.

### 

|  |  |  |
| --- | --- | --- |
| **ImpersonationOption** | **ImpersonateCallerForAllServiceOperations** | **Behavior** |
| Required | n/a | WCF impersonates the caller |
| Allowed | false | WCF does not impersonate the caller |
| Allowed | true | WCF impersonates the caller |
| NotAllowed | false | WCF does not impersonate the caller |
| NotAllowed | true | Disallowed. (An [InvalidOperationException](http://msdn.microsoft.com/en-us/library/system.invalidoperationexception.aspx) is thrown.) |

## Impersonation Level Obtained from Windows Credentials and Cached Token Impersonation

In some scenarios the client has partial control over the level of impersonation the service performs when a Windows client credential is used. One scenario occurs when the client specifies an Anonymous impersonation level. The other occurs when performing impersonation with a cached token. This is done by setting the [AllowedImpersonationLevel](http://msdn.microsoft.com/en-us/library/system.servicemodel.security.windowsclientcredential.allowedimpersonationlevel.aspx) property of the [WindowsClientCredential](http://msdn.microsoft.com/en-us/library/system.servicemodel.security.windowsclientcredential.aspx) class, which is accessed as a property of the generic [ChannelFactory](http://msdn.microsoft.com/en-us/library/ms576132.aspx) class.

|  |
| --- |
| **noteNote:** |
| Specifying an impersonation level of Anonymous causes the client to log on to the service anonymously. The service must therefore allow anonymous logons, regardless of whether impersonation is performed. |

The client can specify the impersonation level as [Anonymous](http://msdn.microsoft.com/en-us/library/system.security.principal.tokenimpersonationlevel.aspx), [Identification](http://msdn.microsoft.com/en-us/library/system.security.principal.tokenimpersonationlevel.aspx), [Impersonation](http://msdn.microsoft.com/en-us/library/system.security.principal.tokenimpersonationlevel.aspx), or [Delegation](http://msdn.microsoft.com/en-us/library/system.security.principal.tokenimpersonationlevel.aspx). Only a token at the specified level is produced, as shown in the following code.

C#

[VB](http://msdn.microsoft.com/en-us/library/ms730088(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

ChannelFactory<IEcho> cf = new ChannelFactory<IEcho>("EchoEndpoint");

cf.Credentials.Windows.AllowedImpersonationLevel =

System.Security.Principal.TokenImpersonationLevel.Impersonation;

The following table specifies the impersonation level the service obtains when impersonating from a cached token.

|  |  |  |  |
| --- | --- | --- | --- |
| **AllowedImpersonationLevel value** | **Service has SeImpersonatePrivilege** | **Service and client are capable of delegation** | **Cached token ImpersonationLevel** |
| Anonymous | Yes | n/a | Impersonation |
| Anonymous | No | n/a | Identification |
| Identification | n/a | n/a | Identification |
| Impersonation | Yes | n/a | Impersonation |
| Impersonation | No | n/a | Identification |
| Delegation | Yes | Yes | Delegation |
| Delegation | Yes | No | Impersonation |
| Delegation | No | n/a | Identification |

## Impersonation Level Obtained from User Name Credentials and Cached Token Impersonation

By passing the service its user name and password, a client enables WCF to log on as that user, which is equivalent to setting the **AllowedImpersonationLevel** property to [Delegation](http://msdn.microsoft.com/en-us/library/system.security.principal.tokenimpersonationlevel.aspx). (The **AllowedImpersonationLevel** is available on the [WindowsClientCredential](http://msdn.microsoft.com/en-us/library/system.servicemodel.security.windowsclientcredential.aspx) and [HttpDigestClientCredential](http://msdn.microsoft.com/en-us/library/system.servicemodel.security.httpdigestclientcredential.aspx) classes.) The following table provides the impersonation level obtained when the service receives user name credentials.

|  |  |  |  |
| --- | --- | --- | --- |
| **AllowedImpersonationLevel** | **Service has SeImpersonatePrivilege** | **Service and client are capable of delegation** | **Cached token ImpersonationLevel** |
| n/a | Yes | Yes | Delegation |
| n/a | Yes | No | Impersonation |
| n/a | No | n/a | Identification |

## Impersonation Level Obtained from S4U-Based Impersonation

|  |  |  |  |
| --- | --- | --- | --- |
| **Service has SeTcbPrivilege** | **Service has SeImpersonatePrivilege** | **Service and client are capable of delegation** | **Cached token ImpersonationLevel** |
| Yes | Yes | n/a | Impersonation |
| Yes | No | n/a | Identification |
| No | n/a | n/a | Identification |

## Mapping a Client Certificate to a Windows Account

It is possible for a client to authenticate itself to a service using a certificate, and to have the service map the client to an existing account through Active Directory. The following XML shows how to configure the service to map the certificate.

XML

<behaviors>

<serviceBehaviors>

<behavior name="MapToWindowsAccount">

<serviceCredentials>

<clientCertificate>

<authentication mapClientCertificateToWindowsAccount="true" />

</clientCertificate>

</serviceCredentials>

</behavior>

</serviceBehaviors>

</behaviors>

The following code shows how to configure the service.

// Create a binding that sets a certificate as the client credential type.

WSHttpBinding b = new WSHttpBinding();

b.Security.Message.ClientCredentialType = MessageCredentialType.Certificate;

// Create a service host that maps the certificate to a Windows account.

Uri httpUri = new Uri("http://localhost/Calculator");

ServiceHost sh = new ServiceHost(typeof(HelloService), httpUri);

sh.Credentials.ClientCertificate.Authentication.MapClientCertificateToWindowsAccount = true;

## Delegation

To delegate to a back-end service, a service must perform Kerberos multi-leg (SSPI without NTLM fallback) or Kerberos direct authentication to the back-end service using the client’s Windows identity. To delegate to a back-end service, create a [ChannelFactory](http://msdn.microsoft.com/en-us/library/ms576132.aspx) and a channel, and then communicate through the channel while impersonating the client. With this form of delegation, the distance at which the back-end service can be located from the front-end service depends on the impersonation level achieved by the front-end service. When the impersonation level is [Impersonation](http://msdn.microsoft.com/en-us/library/system.security.principal.tokenimpersonationlevel.aspx), the front-end and back-end services must be running on the same machine. When the impersonation level is [Delegation](http://msdn.microsoft.com/en-us/library/system.security.principal.tokenimpersonationlevel.aspx), the front-end and back-end services can be on separate machines or on the same machine. Enabling delegation-level impersonation requires that Windows domain policy be configured to permit delegation. For more information about configuring Active Directory for delegation support, see [Enabling Delegated Authentication](http://go.microsoft.com/fwlink/?LinkId=99690).

|  |
| --- |
| **noteNote:** |
| When a client authenticates to the front-end service using a user name and password that correspond to a Windows account on the back-end service, the front-end service can authenticate to the back-end service by reusing the client’s user name and password. This is a particularly powerful form of identity flow, because passing user name and password to the back-end service enables the back-end service to perform impersonation, but it does not constitute delegation because Kerberos is not used. Active Directory controls on delegation do not apply to user name and password authentication. |

### Delegation Ability as a Function of Impersonation Level

|  |  |  |
| --- | --- | --- |
| **Impersonation level** | **Service can perform cross-process delegation** | **Service can perform cross-machine delegation** |
| [Identification](http://msdn.microsoft.com/en-us/library/system.security.principal.tokenimpersonationlevel.aspx) | No | No |
| [Impersonation](http://msdn.microsoft.com/en-us/library/system.security.principal.tokenimpersonationlevel.aspx) | Yes | No |
| [Delegation](http://msdn.microsoft.com/en-us/library/system.security.principal.tokenimpersonationlevel.aspx) | Yes | Yes |

The following code example demonstrates how to use delegation.

C#

[VB](http://msdn.microsoft.com/en-us/library/ms730088(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-7)

public class HelloService : IHelloService

{

[OperationBehavior(Impersonation = ImpersonationOption.Required)]

public string Hello(string message)

{

WindowsIdentity callerWindowsIdentity = ServiceSecurityContext.Current.WindowsIdentity;

if (callerWindowsIdentity == null)

{

throw new InvalidOperationException

("The caller cannot be mapped to a Windows identity.");

}

using (callerWindowsIdentity.Impersonate())

{

EndpointAddress backendServiceAddress = new EndpointAddress("http://localhost:8000/ChannelApp");

// Any binding that performs Windows authentication of the client can be used.

ChannelFactory<IHelloService> channelFactory = new ChannelFactory<IHelloService>(new NetTcpBinding(), backendServiceAddress);

IHelloService channel = channelFactory.CreateChannel();

return channel.Hello(message);

}

}

}

### How to Configure an Application to Use Constrained Delegation

Before you can use constrained delegation, the sender, receiver, and the domain controller must be configured to do so. The following procedure lists the steps that enable constrained delegation. For details about the differences between delegation and constrained delegation, see the portion of [Windows Server 2003 Kerberos Extensions](http://go.microsoft.com/fwlink/?LinkId=100194) that discusses constrained discussion.

1. On the domain controller, clear the **Account is sensitive and cannot be delegated** check box for the account under which the client application is running.
2. On the domain controller, select the **Account is trusted for delegation** check box for the account under which the client application is running.
3. On the domain controller, configure the middle tier computer so that it is trusted for delegation, by clicking the **Trust computer for delegation** option.
4. On the domain controller, configure the middle tier computer to use constrained delegation, by clicking the **Trust this computer for delegation to specified services only** option.

For more detailed instructions about configuring constrained delegation, see the following topics on MSDN:

* [Troubleshooting Kerberos Delegation](http://go.microsoft.com/fwlink/?LinkId=36724)
* [Kerberos Protocol Transition and Constrained Delegation](http://go.microsoft.com/fwlink/?LinkId=36725)

# identity Element (ASP.NET Settings Schema)

**.NET Framework 4**

[Other Versions](javascript:;)



* [.NET Framework 3.5](http://msdn.microsoft.com/en-us/library/72wdk8cc(d=printer,v=vs.90).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/72wdk8cc(d=printer,v=vs.85).aspx)
* [.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/72wdk8cc(d=printer,v=vs.80).aspx)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/72wdk8cc(d=printer,v=vs.71).aspx)

Configures the identity of the Web application. This element can be declared at any level in the configuration file hierarchy.

|  |
| --- |
| **NoteNote** |
| The example syntax in this topic includes a password to demonstrate how the syntax works. In your applications, we recommend that you use a strategy to secure passwords. |

[<configuration> Element](http://msdn.microsoft.com/en-us/library/5x77e536(v=vs.100).aspx)  
  [system.web Element (ASP.NET Settings Schema)](http://msdn.microsoft.com/en-us/library/dayb112d(v=vs.100).aspx)  
    [identity Element (ASP.NET Settings Schema)](http://msdn.microsoft.com/en-us/library/72wdk8cc(v=vs.100).aspx)

<identity impersonate="true|false"

userName="domain\username"

password="<secure password>"/>

[Attributes and Elements](javascript:void(0))

The following sections describe attributes, child elements, and parent elements.

#### Attributes

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| Impersonate | Required attribute.  Specifies whether client [impersonation](http://msdn.microsoft.com/en-us/library/xh507fc5(v=vs.100).aspx) is used on each request.  This attribute can be one of the following possible values.  ValueDescription  falseSpecifies that client impersonation is not used.  trueSpecifies that client impersonation is used. |
| Password | Optional attribute.  Specifies the password to use, if the impersonate attribute is true.  For information about storing encrypted worker process credentials in the registry, see the userName attribute. |
| userName | Optional attribute.  Specifies the user name to use, if the impersonate attribute is true.  This attribute and the password attribute are stored in clear text in the configuration file. Although Microsoft Internet Information Services (IIS) will not transmit .config files in response to a user agent request, .config files can be read by other means. For example, by an authenticated user with the proper credentials on the domain that contains the server. For security reasons, the identity attribute supports storing encrypted userName and password attributes in the registry. The credentials must be in REG\_BINARY format and encrypted by the Microsoft Windows 2000 and Windows XP Data Protection API (DPAPI) encryption functions.  For more information, see "Remarks" and "Example," later in this topic. |

#### Child Elements

None.

#### Parent Elements

|  |  |
| --- | --- |
| **Element** | **Description** |
| configuration | Specifies the root element in every configuration file that is used by the common language runtime and the .NET Framework applications. |
| system.web | Specifies the root element for the ASP.NET configuration section. |

[Remarks](javascript:void(0))

To encrypt the user name and password and store the user name and password in the registry, set the userName and password attributes as follows.

userName="registry:HKLM\Software\AspNetProcess,Name"password="registry:HKLM\Software\AspNetProcess,Pwd"

The portion of the string after the keyword registry and before the comma indicates the name of the registry key that ASP.NET opens. The portion after the comma contains a single string value name from which ASP.NET reads the credentials. The comma is required and the credentials must be stored in the HKLM hive. If the configuration format is incorrect, ASP.NET will not launch the worker process and will follow the current account creation failure code path.

The credentials must be in REG\_BINARY format, containing the output of a call to the Windows API function CryptProtectData. You can create the encrypted credentials and store them in the registry with Aspnet\_setreg.exe, which uses CryptProtectData to accomplish the encryption. To download Aspnet\_setreg.exe, along with the Microsoft Visual C++ source code and documentation, go to the [ASP.NET Web site](http://go.microsoft.com/fwlink/?linkid=37105) and search for aspnet\_setreg.

You should configure access to the key that is storing the encrypted credentials so that access is provided only to Administrators and SYSTEM. Because the key will be read by the ASP.NET process that is running as SYSTEM, you should set the following permissions:

* Administrators:F
* SYSTEM:F
* CREATOR OWNER:F
* ProcessAccount:R

This provides two lines of defense to help protect the data, as follows:

* The ACL permissions require the identity that is accessing the data to be Administrator.
* An attacker must run code on the server (the CryptUnprotectData API) to recover the credentials for the account.

### Default Configuration

The following default identity element is not explicitly configured in the Machine.config file or in the root Web.config file. However, it is the default configuration that is returned by application.

<identity impersonate="false" userName="" password="" />

[Element Information](javascript:void(0))

|  |  |
| --- | --- |
| **Configuration section handler** | [System.Web.Configuration.IdentitySection](http://msdn.microsoft.com/en-us/library/system.web.configuration.identitysection(v=vs.100).aspx) |
| **Configuration member** | [SystemWebSectionGroup.Identity](http://msdn.microsoft.com/en-us/library/system.web.configuration.systemwebsectiongroup.identity(v=vs.100).aspx) |
| **Configurable locations** | Machine.config  Root-level Web.config  Application-level Web.config  Virtual or physical directory–level Web.config |
| **Requirements** | Microsoft Internet Information Services (IIS) 5.0, 5.1, or 6.0  The .NET Framework version 1.0, 1.1, or 2.0  Microsoft Visual Studio 2003 or Visual Studio 2005 |

**ASP.NET Impersonation**

**Visual Studio .NET 2003**

Another important security feature is the ability to control the identity under which code is executed. Impersonation is when ASP.NET executes code in the context of an authenticated and authorized client. By default, ASP.NET does not use impersonation and instead executes all code using the same user account as the ASP.NET process, which is typically the ASPNET account. This is contrary to the default behavior of ASP, which uses impersonation by default. In Internet Information Services (IIS) 6, the default identity is the NetworkService account.

**Note**   Impersonation can significantly affect performance and scaling. It is generally more expensive to impersonate a client on a call than to make the call directly.

Using impersonation, ASP.NET applications can optionally execute the processing thread using the identity of the client on whose behalf they are operating. You usually use impersonation for resource access control. Delegation is a more powerful form of impersonation and makes it possible for the server process to access remote resources while acting as the client. For more information, see [ASP.NET Delegation](http://msdn.microsoft.com/en-us/library/aa291350(v=vs.71).aspx).

**Note**   Impersonation is local to a particular thread. When code changes threads, such as when using thread pooling, the new thread executes using the process identity by default. When impersonation is required on the new thread, your application should save the security token ([WindowsIdentity.Token Property](http://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity.token(v=vs.71).aspx)) from the original thread as part of the state for the completion thread.

If you enable impersonation, ASP.NET can either impersonate the authenticated identity received from IIS or one specified in the application's Web.config file. You have the following three options when configuring impersonation:

* **Impersonation is disabled**. This is the default setting. For backward compatibility with ASP, you must enable impersonation and change the ASP.NET process identity to use the Local System account. In this instance, the ASP.NET thread runs using the process token of the application worker process regardless of which combination of IIS and ASP.NET authentication is used. By default, the process identity of the application worker process is the ASPNET account. For more information, see [ASP.NET Process Identity](http://msdn.microsoft.com/en-us/library/aa291339(v=vs.71).aspx).

<identity impersonate="false" />

* **Impersonation enabled**. In this instance, ASP.NET impersonates the token passed to it by IIS, which is either an authenticated user or the anonymous Internet user account (IUSR\_*machinename*).

<identity impersonate="true" />

* **Impersonation enabled for a specific identity**. In this instance, ASP.NET impersonates the token generated using an identity specified in the Web.config file.

<identity impersonate="true"

userName="domain\user"

password="password" />

If the application resides on a UNC share, ASP.NET always impersonates the IIS UNC token to access that share unless a configured account is used. If you provide an explicitly configured account, ASP.NET uses that account in preference to the IIS UNC token.

You should exercise care when using impersonation because it makes it possible for an application to potentially process code using permissions not anticipated by the application designer. For example, if your application impersonates an authenticated intranet user, that application possesses administrative privileges when impersonating a user with those privileges. Likewise, if the impersonated user possesses more restrictive permissions than anticipated, the user may not be able to use the application.

# ASP.NET Delegation

**Visual Studio .NET 2003**

Impersonation enables ASP.NET to execute code and access resources in the context of an authenticated and authorized user, but only on the server where ASP.NET is running. To access resources located on another computer on behalf of an impersonated user requires authentication delegation (or delegation for short). You can think of delegation as a more powerful form of impersonation, as it enables impersonation across a network. For more information, see [ASP.NET Impersonation](http://msdn.microsoft.com/en-us/library/aa292118(v=vs.71).aspx).

For delegation to work, ASP.NET must be able to authenticate against the remote server using the credentials of the client you wish to delegate. However, there are numerous factors that determine if delegation can occur, such as Internet Information Services (IIS) authentication scheme, ASP.NET process identity, and the operating systems of the machines involved.

Misuse of delegation could make the network vulnerable to sophisticated attacks using Trojan horse programs that impersonate incoming clients and use their credentials to gain access to network resources.

### Authentication Schemes

On Windows 2000, the following authentication schemes impact delegation of impersonated clients as follows:

#### Anonymous

When using impersonation with an anonymous client, ASP.NET impersonates the account specified by IIS for anonymous access. By default, this is the IUSR\_*machinename* account with a password controlled by IIS, which cannot be delegated.

However, if IIS is configured to use a local account that is identical (including password) to a local account on the remote machine, delegation is possible. In addition, if IIS is configured to use a domain account that has access to the remote machine, delegation is also possible.

#### Basic

When using Basic authentication, delegation is possible if the account is a local account that is identical (including password) to a local account on the remote machine. In addition, delegation is also possible if the account is a domain account that has access to the remote machine.

#### Digest

Digest authentication does not permit delegation since IIS does not posses any credentials for the client with which to authenticate against the remote machine.

#### Integrated Windows Authentication

When Internet Explorer attempts to access a protected resource, IIS sends two WWW-Authenticate headers to the browser, Negotiate and NTLM. The Negotiate header is only sent by IIS running on Windows 2000 or later. This header indicates that IIS supports the Negotiate protocol, which enables a negotiation to occur between Internet Explorer and IIS on whether to use Kerberos or NTLM authentication. IIS uses Kerberos if both the client (Internet Explorer 5.0 and later) and server (IIS 5.0 and later) are running Windows 2000 or later, and both are members of the same domain or trusted domains. Otherwise, the server defaults to using NTLM.

Since NTLM authenticates the user for IIS without providing the user's credentials to IIS, IIS cannot delegate that user's credentials to a remote machine.

When used in conjunction with Kerberos v5 authentication, IIS can delegate security credentials among computers running Windows 2000 or later that are trusted and configured for delegation.

#### Client Certificate Mapping

Client Certificate Mapping supports delegation in some instances. That is, if you use IIS mapping (many-to-one) and the mapped account is identical (including password) to a local account on the remote machine, delegation is possible. In addition, delegation is possible if IIS maps the certificate to a domain account that has access to the remote machine. In both cases IIS stores credentials for the account in question.

However, if using Active Directory mapping (one-to-one), delegation is not possible. This is because Active Directory holds the certificate for each account, not IIS.

### ASP.NET Process Identity

If impersonation is not configured for an ASP.NET application, the ASP.NET process identity determines which account to use for delegation when accessing remote machines as follows:

#### Machine (ASPNET)

When using <processModel userName="machine">, which is the default setting, ASP.NET attempts to delegate the ASPNET local user account. As this account does not possess any network credentials, to the network it appears as the Windows anonymous account (NT AUTHORITY\ANONYMOUS LOGON).

**Note**   You should not confuse the Windows anonymous account (NT AUTHORITY\ANONYMOUS LOGON) with the anonymous access account used by IIS to provide anonymous Web access (IUSR\_*machinename*). The Windows anonymous account is used by the operating system when an identity cannot be authenticated.

#### System

When using <processModel userName="SYSTEM">, ASP.NET attempts to delegate the account used for running IIS, which by default is the LocalSystem account (NT AUTHORITY\SYSTEM). Starting with Windows 2000, this account possesses the network credentials associated with the machine account in the domain of which it is a member (*domainname\machinename*).

#### Configured Identity

When you configure <processModel> to use a particular account as the process identity, ASP.NET attempts to delegate that account. If it is a local account that is identical (including password) to a local account on a remote machine, delegation is possible. If such an account does not exist on the remote machine, to the network it appears as the Windows anonymous account (NT AUTHORITY\ANONYMOUS LOGON). In addition, delegation is also possible if the account is a domain account that has access to the remote machine, in which case it uses the domain network identity of that account.

#### NetworkService

In Internet Information Services (IIS) 6.0, the default identity is the NetworkService account. It behaves the same as the System account. This account possesses the network credentials associated with the machine account (*domainname\machinename*) in the domain of which it is a member.

### Delegation Alternatives

There are several methods for accessing remote resources when you cannot delegate using your chosen authentication scheme, such as:

* Create a serviced component that accesses the remote resource using a configured identity, and then access that component from ASP.NET. For more information, see [Serviced Component Overview](http://msdn.microsoft.com/en-us/library/aa720610(v=vs.71).aspx).
* Call the LogonUser platform API to create a logon session that you can impersonate to access the remote resource. The LogonUser platform API requires the SE\_TCB\_NAME privilege (the "Act as part of the operating system" privilege) on Windows 2000 but not on Windows XP and later. For more information, see [WindowsIdentity.Impersonate Method](http://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity.impersonate(v=vs.71).aspx).
* Use a null session share, which enables anonymous access to UNC shares. For more information, see "HOW TO: Enable Null Session Shares on a Windows 2000-Based Computer (Q289655)" on the Microsoft Knowledge Base Web site ([http://support.microsoft.com/default.aspx?scid=kb;EN-US;q289655](http://support.microsoft.com/default.aspx?scid=kb;en-us;q289655)).

**Security Note**   Using null session shares to grant access to a shared resource is not a recommended practice. It is mentioned here to recognize this capability and to warn against its use. If you configure a shared resource in this manner, the resource is not secure as it grants access to anyone requesting it.