Code Access Security

© 2006 newtelligence AG

Revision: 1.0.1

Author: Michael Willers (michaelw@newtelligence.com)

newtelligence AG, Gilleshütte 99, 41352 Korschenbroich

1 Introduction 1
2 Concept of Code Access Security 2
2.1 Evidence 5
2.2 Codegroup 5
2.3 Permissions 6
2.4 Permission Sets 6
2.5 Policies 7
2.6 Security Checks 10
2.6.1 Declarative vs. Imperative Security Checks 10
2.6.2 Demands 10
2.6.3 Requests 11
2.6.4 Overrides 12
3 Overview on Partial Trust 14
4 List of Predefined Permissions and its Parameters 16
4.1 Permissions represented via Configuration Console 16
4.1.1 System.Data.SqlClient.SqlClientPermission 16
4.1.2 System.Diagnostics.EventLogPermission 17
4.1.3 System.Diagnostics.PerformanceCounterPermission 17
4.1.4 System.DirectoryServices.DirectoryServicesPermission 18
4.1.5 System.Drawing.Printing.PrintingPermission 18
4.1.6 System.Messaging.MessageQueuePermission 18
4.1.7 System.Net.DnsPermission 19
4.1.8 System.Net.SocketPermission 19
4.1.9 System.Net.WebPermission 20
4.1.10 System.Security.Permissions.EnvironmentPermission 21
4.1.11 System.Security.Permissions.FileDialogPermission 22
4.1.12 System.Security.Permissions.FileIOPermission 22
4.1.13 System.Security.Permissions.IsolatedStorageFilePermission 24
4.1.14 System.Security.Permissions.ReflectionPermission 25
4.1.15 System.Security.Permissions.RegistryPermission 26
4.1.16 System.Security.Permissions.SecurityPermission 27
4.1.17 System.Security.Permissions.StorePermission 28
4.1.18 System.Security.Permissions.UIPermission 28
4.1.19 System.ServiceProcess.ServiceControllerPermission 29
4.2 Permissions not represented in Configuration Console 29
4.2.1 System.Configuration.ConfigurationPermission 29
4.2.2 System.Data.Common.DBDataPermission 30
4.2.3 System.Data.Odbc.OdbcPermission 30
4.2.4 System.Data.OleDb.OleDbPermission 30
4.2.5 System.Data.OracleClient.OraclePermission 31
4.2.6 System.Net.Mail.SmtpPermission 31
4.2.7 System.Net.NetworkInformation.NetworkInformationPermission 31
4.2.8 System.Security.Permissions.DataProtectionPermission 32
4.2.9 System.Security.Permissions.GacIdentityPermission 32
4.2.10 System.Security.Permissions.KeyContainerPermission 32
4.2.11 System.Security.Permissions.PublisherIdentityPermission 33
4.2.12 System.Security.Permissions.ResourcePermissionBase 34
4.2.13 System.Security.Permissions.SiteIdentityPermission 34
4.2.14 System.Security.Permissions.StrongNameIdentityPermission 35
4.2.15 System.Security.Permissions.UrlIdentityPermission 36
4.2.16 System.Security.Permissions.ZoneIdentityPermission 36
4.2.17 System.Transactions.DistributedTransactionPermission 37
4.2.18 System.Web.AspNetHostingPermission 37

1 Introduction

It seems to be a way of no return. We don’t want to abandon working online, living online and being online. But the more machines and devices are connected to the global hub “internet” the more we dependent on the fact of being “always-on”. Along with shifting analog content to the digital world, interesting information is hold available to be remotely snatched by the bad guys. Nowadays this happens most frequently by mobile code being actively downloaded from Internet and executed on the local machine. Majority of Microsoft Windows users still do work with administrative rights, so this evil code could initiate hardly predictable activities.

Code Access Security (CAS) is a security model of the Microsoft .NET-Framework's Common Language Runtime and allows secure code execution by running in a sandboxed environment. The CLR grants access to system resources based on security policies. Policies are applied by evidences of the code, for instance from where the code is loaded or who is the publisher of the code.

CAS provides fine levels of granularity in restricting access to system resources. The .NET Framework delivers about 40 permissions with diverse parameters, a dozen code groups and 7 evidences.

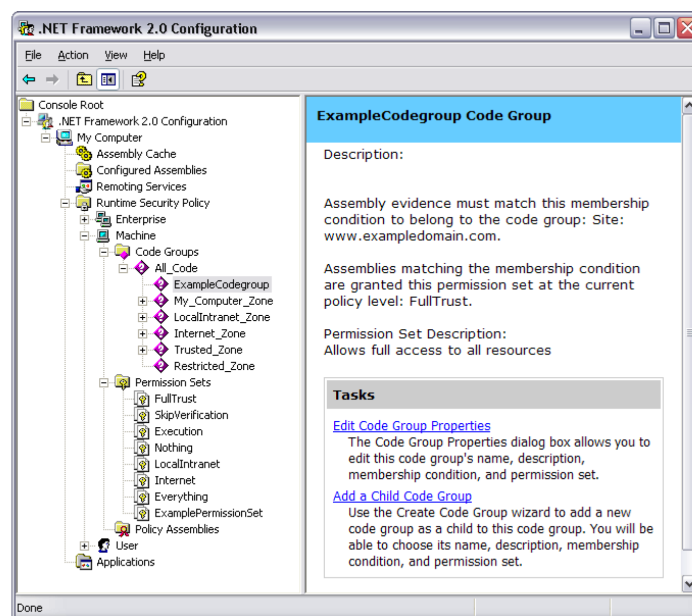
The goal of this document is to provide an overview on CAS in .NET Framework 2.0 and collect information about it currently spread on MSDN and several weblogs throughout the community.

2 Concept of Code Access Security

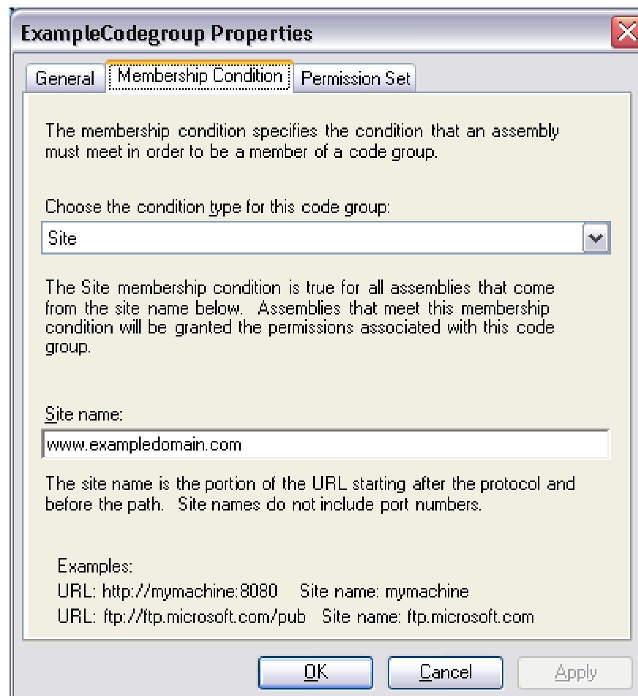
Security in .NET is based on two different approaches: Role Based Security (RBS) and Code Access Security (CAS). Whereas RBS grants access to resources according to authenticated users, CAS uses code characteristics as a basis to decide whether to allow or restrict access to a resource. Code Access Security is the solution to secure mobile code, downloaded from other than the local machine. Thereby all code is executed in a protected environment called the sandbox. The CLR grants or denies accessing system resources via security policies. A policy is linked with characteristics of the code to be downloaded and executed. The technical term for characteristics here is “evidence”.

Example:
Imagine code to be downloaded from a website “http://www.exampledomain.com/statistics”. Evidence could be the sitename “www.exampledomain.com”. Permission could be “do not allow using my machine’s printer”. A policy could be: “Code with the upper evidence will get the specified permission”. That means: Code from site “www.exampledomain.com” is not allowed to use my machine’s printer. This security behaviour is expressed within a codegroup “ExampleCodegroup”. The following screenshots illustrate this scenario in the .NET configuration console.

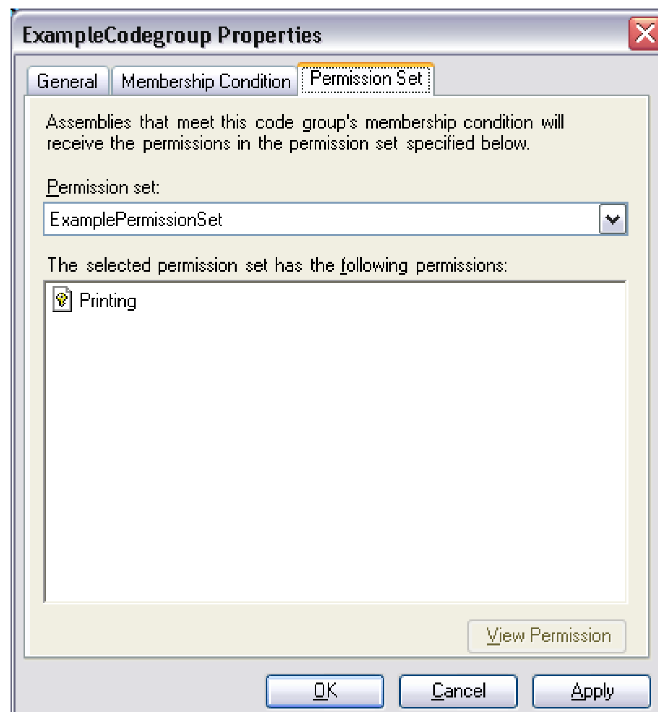
Open the configuration console via “Start” -> “Control Panel” -> “Administrative Tools” -> “Microsoft .NET Framework 2.0 Configuration”. Collapse the nodes “My Computer” -> “Runtime Security Policy” -> “Machine” -> “Code Groups” -> “All_Code”:



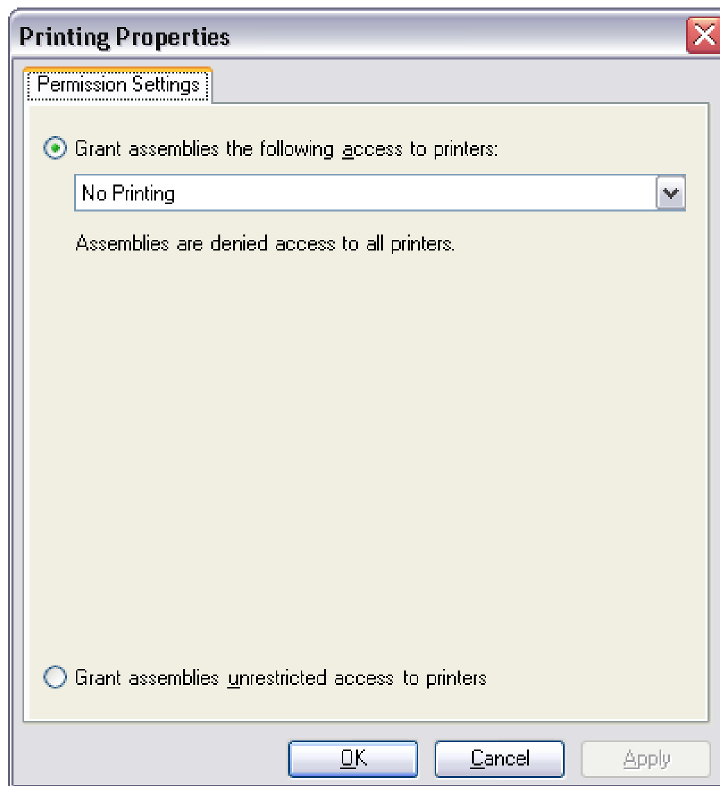
Here we created a custom codegroup “ExampleCodegroup”. The properties show the membership condition mapping assembly evidence and a permission set to the codegroup. Here the evidence for the membership condition is the sitename with value “www.exampledomain.com”:



The permission set mapped to our codegroup “ExampleCodegroup” is called “ExamplePermissionSet” and contains a single permission “Printing”:

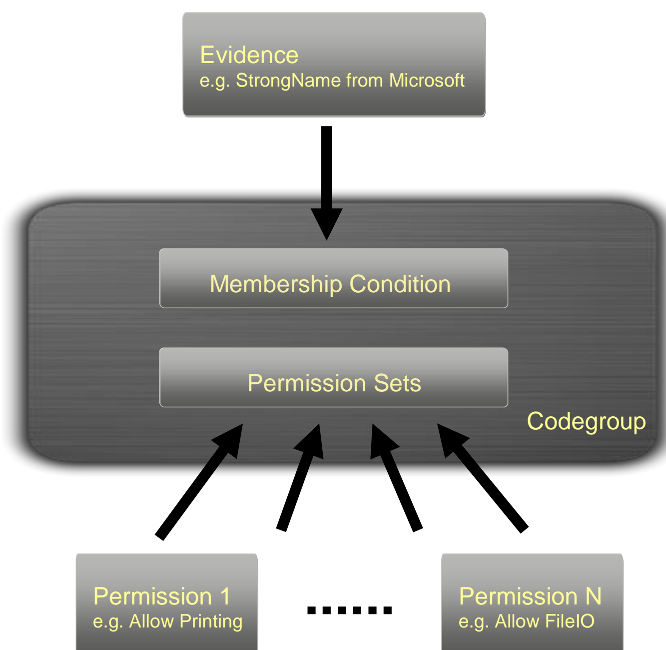


Collapse the nodes “My Computer” -> “Runtime Security Policy” -> “Machine” -> “Permission Sets”. Here we create a permission set called “ExamplePermissionSet”. It contains only the single permission “Printing”:



Before downloading the assembly, the CLR collects all evidences about it. The CLR transfers this data to the securitypolicy module, which designates permissions the assembly according to the given evidence. This mapping between evidence and permission is expressed within a codegroup. So a codegroup contains all information defining the actions that are granted resp. denied to a somehow specified assembly.

Strictly speaking, evidence is mapped to a permissionset which is a collection of permissions. This mapping is expressed by a membership condition and consolidated within a codegroup. Note that a codegroup contains just a single evidence but may contain several permissions. The following chart illustrates these relations:



2.1 Evidence

The first problem that arises when defining a security model on "block of codes" is to define the entity to apply security checks on, since code has nothing like a unique "user identity" (aka principal) as we can find in role based security models with for example NTLM, Kerberos.

CAS identifies an assembly on its evidence, that is, a list of property name—property value pair that describes it. The CLR comes with a predefined set of evidences. They can be divided in origin-related evidences on one side and content-related evidences on the other:

Origin-related Evidence

- Application Directory – Application Directory membership condition is true for all assemblies in the same directory or in a child directory of the running application
- GAC – GAC membership condition is true for all assemblies which are installed in the GAC
- Site – Site membership condition is true for all assemblies that originate from the specified site. E.g. “www.exampledomain.com”
- URL – Url membership condition is true for all assemblies that originate from the specified URL. E.g. “http://www.exampledomain.com/pub/exampleapp.dll” or “ftp://ftp.exampledomain.com/pub/*”
- Zone – Zone membership condition is true for all assemblies that originate from the specified zone. E.g. Trusted Sites. Mapping Urls to zones can be done via "Start" -> "Control Panel" -> "Internet Options" -> Security Tab -> Select a Zone -> "Sites"

Content-related Evidence

- Hash – Hash membership condition is true for all assemblies with a hash that matches a specified hash based on MD5 or SHA1 algorithm
- Publisher – Publisher membership condition is true if an assembly is digitally signed with a certificate that matches a specified certificate
- StrongName – Strong Name membership condition is true for all assemblies with a strong name that matches a specified strong name.

Note that not all the assemblies have the same set of evidence since it depends on the host loading the assembly. For instance an assembly loaded from the file system will have no site evidence associated with it.
Know that some evidences are stronger then other, depending on how easy they can be tampered. For instance, the strong name evidence of an assembly is stronger than its site evidence (Websites and DNS can be hacked). It’s also possible to define custom application or system based evidence types.

2.2 Codegroup

Codegroups are logical entities that group code according to specified condition memberships. Code from “http://www.exampledomain.com/” may belong to one code group, code containing a specific strong name can belong to another code group and code from a specific assembly can belong to another code group.

By default the .NET framework comes up with 8 built-in codegroups located at the machine policy level (Policy levels are described in the “Policy” chapter). The following list shows those codegroups and the permission sets they receive by default.

Codegroup	Description	Built-In Permission Set
My Computer Zone	Code from the local computer	Full Trust
Microsoft Strong Name	Code signed with the Microsoft strong name	
EMCA Strong Name	Code signed with the EMCA strong name	
Local Intranet Zone	Code from a local network	Local Intranet (“medium trust”)

<table>
<tr><td>Internet Zone</td><td>Code from the internet</td><td rowspan="2">Internet (“low trust”)</td></tr>
<tr><td>Trusted Zone</td><td>Code from trusted sites in Internet Settings</td></tr>
<tr><td>All Code</td><td>All managed code</td><td rowspan="2">Nothing (“no trust”)</td></tr>
<tr><td>Restricted Zone</td><td>Code from restricted sites</td></tr>
</table>

The default policy works on the premise that code installed on your machine is more trusted than code that is loaded from the network. And of course there is a provision that ensures that assemblies in the .NET Framework itself are always fully trusted. After all, some piece of code has to be trusted to implement this whole security infrastructure.

To keep things simple, Microsoft's default policy grants permissions based on zone evidence. The MyComputer zone is granted full trust. This represents locally installed code such as applications you've installed from a CD- or DVD-ROM, or programs that you have manually downloaded and saved to your disk before running.

Next is the LocalIntranet zone. If you don't explicitly list which Web sites are part of this zone, it's simply defined as any domain name without a "." in the name (in other words it's a NETBIOS hostname as opposed to a DNS name). For example, “http://xyz” would be considered part of the LocalIntranet zone by default, while “http://www.microsoft.com” would normally drop into the Internet bucket because of the dots in the name, as would http://207.46.250.119. Note that the decimal equivalent http://3475962487 will not be accepted; you can read more about the "Dotless-IP Address" bug in Michael Howard's book, Writing Secure Code, Second Edition.

Default security policy assigns what you might call a "medium trust" permission set to assemblies in the LocalIntranet zone. This means code from your local network will normally run with partial trust (see chapter on “Partial Trust”). If you've ever tried running a managed executable from a shared drive, even one on your local machine (like z:\MySmartClient.exe) you may have run into a SecurityException or two, because the zone you're running from is no longer the MyComputer zone.

Internet Explorer defines a couple of zones designed for customization by system administrators: the Trusted Sites and Restricted Sites zones. The former is granted a "low trust" permission set, and the latter is not trusted at all, which means managed code from a restricted site will not run by default.

And finally, the Internet zone is the bucket into which all URLs fall if they can't be sorted into any of the other zones. The default assignment for this zone has had a history of change, oscillating between low and no trust, but as of version 1.1 of the .NET Framework, it has stabilized and is mapped to the low trust permission set by default.

See chapter “Policy” for information about how permissions are resolved in case an assembly is mapped to more than one codegroup.

2.3 Permissions

The .NET Framework version 2.0 now defines a whole host of about 40 built-in permissions, protecting everything from file and database access to thread suspension and resumption. Just like evidence, each permission is represented as a class, and you can define custom permission classes if the need arises. See chapters “List of Predefined Permissions and its Parameters” and “Experiences on using Permissions” for details.

2.4 Permission Sets

Permissions are grouped by permission sets. A permission set can include any number of permissions. Similar to permission there is a bunch of predefined permission sets which can be used within a codegroup.

- FullTrust – Grants unrestricted permissions to system resources
- SkipVerification – Grants permission to skip security verifications

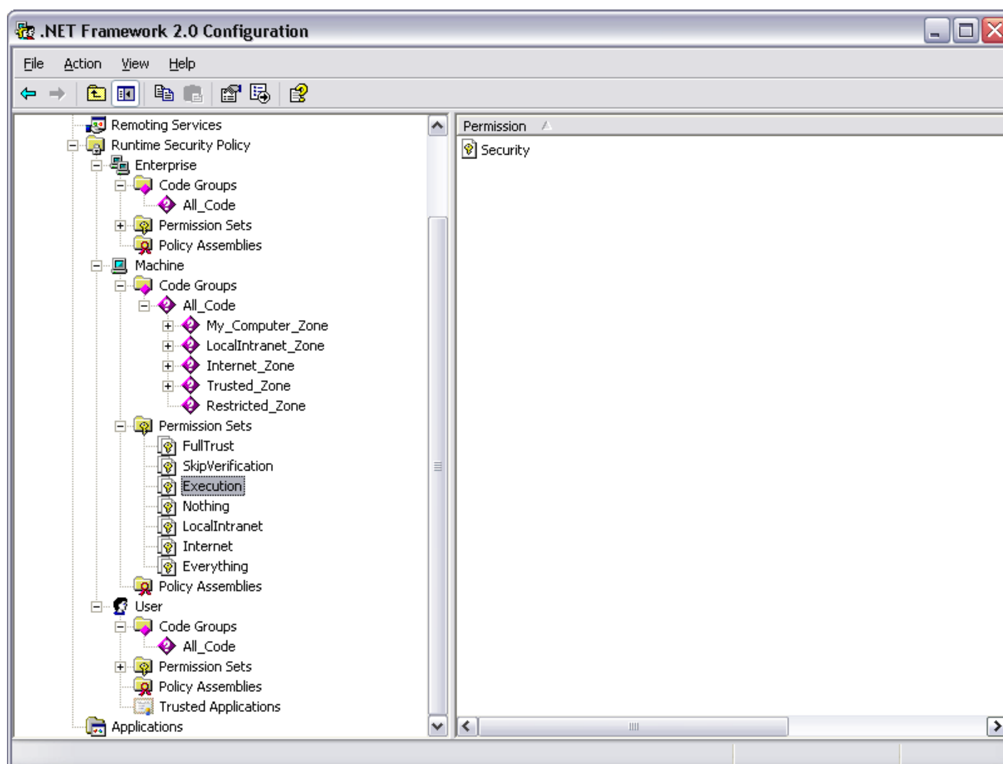
- Execution – Grants permission to execute code
- Nothing – Denies permission to all resources
- LocalIntranet – Grants default permission
- Internet – Grants permissions associated with Internet applications
- Everything – Grants all permissions but does not bypass security verification

Looking at the permission set “FullTrust” and “Everything” it is not quite obvious what the difference between those two is:

- “FullTrust” grants unlimited access to all resources available
- “Everything” grants unlimited access to all resources protected by a permission. So if you create a custom permission and still want to support the default permission sets, make sure to add your permission with an “grant unrestricted access” to permission set “Everything”

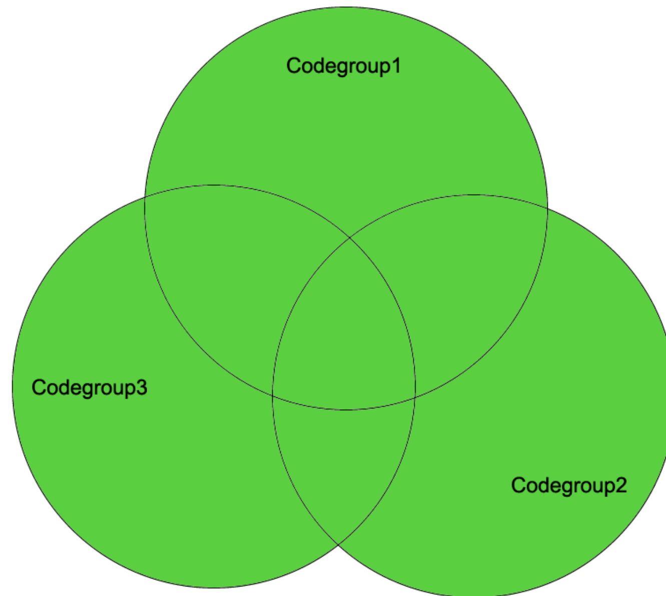
2.5 *Policies*

Security policy is the configurable set of rules that the CLR follows when determining the permissions to grant to code. It is hosting all the defined codegroups in a hierarchical structure. The .NET framework comes with three different security policies: Enterprise, Machine, User. Additionally a host can define application domain-level policy by calling the AppDomain.SetAppDomainPolicy method on the System.AppDomain class. The machine and user policy levels are intended for policy changes that machine administrators and machine users may wish to undertake. They are scoped to be either machine- or user-context bound. The enterprise policy level, however, is intended to carry enterprise-wide security policy configuration state. The levels are organised in the hierarchy shown below:

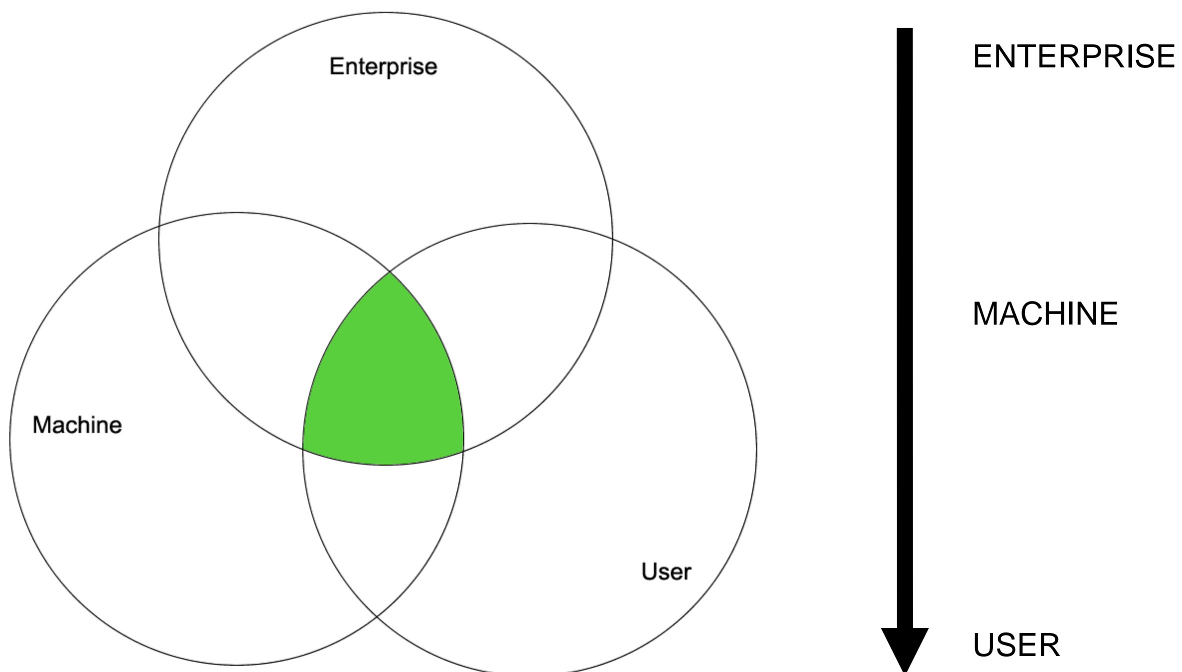


Within each policy level is a set of code groups defining policy at that level. Policy is resolved by evaluating policy for each level, using code groups, and then resolving the final policy, using policy levels.
Inside a policy level is a set of hierarchically organized code groups. Policy resolution begins at the top of the code group hierarchy, checking assembly evidence with the membership condition. If the assembly meets the membership condition of a given code group, it adds that code group's permission set to its set of permissions for that policy level. Additionally, when a membership condition is met, .NET evaluates code groups lower in the hierarchy. CAS policy traverses the entire code group hierarchy, gathering a set of permissions for every membership condition match within that policy level. The grant set of permissions belonging to that policy level is the union of all permissions for which a membership condition was met. The permissions are unified because of the fact that a single assembly could be mapped to more than one

codegroup (e.g. Zone – Local Machine and a strong name) and so also more than one permissionset. The following figure illustrates the permission grant set, colored in green, of an arbitrary policy level with three code groups whose membership conditions matched evidence on an assembly.



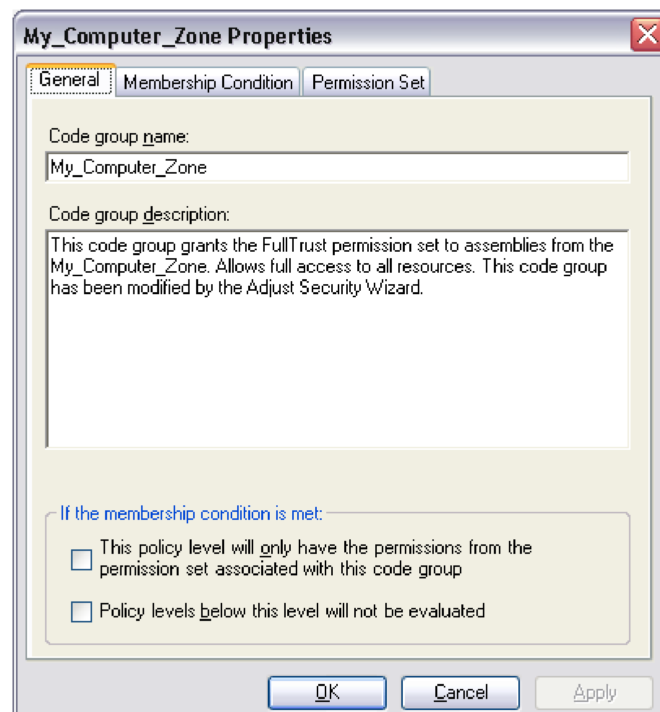
Once .NET computes the grant set for each policy level, the final set of permissions must be resolved. .NET determines the final grant set by calculating the intersection of all the policy levels. In the following figure, the green portion shows the intersection of the permissions granted of the User, Machine, and Enterprise policy levels, which is the final permission set granted to an assembly. Although the AppDomain policy level is not shown in the figure, it also intersects with the other three policy levels.



The default CAS policy configuration gives “All_Code” code group “FullTrust” permissions at the enterprise and user policy levels. This effectively makes the machine policy level control CAS policy a default configuration. This is important to pay attention to because accidentally applying policy to enterprise or user can break a policy and would be hard to find and fix, unless that is what you explicitly want to do.

Codegroups are evaluated by starting on enterprise level, going via machine level down to user level. Permissions denied on higher levels cannot be granted on the lower ones. E.g. you can have more restricted policy on the machine level but you cannot provide more permissions to your machine level than those available from the enterprise level. Now consider a system administrator who needs to install an application that must run on every workstation in the company. He would set the policy for the assemblies belonging to that application at enterprise policy level. This means that settings at the machine and user policy levels will not interfere the policy set at enterprise policy level. This would have the drawback that single policy changes needed to be done on single machines accessing the assembly cannot be done without changing and completely redeploying the enterprise policy. So for setting up custom policies the rule-of-thumb reads as follows: Don't touch "FullTrust" permission at enterprise and user level and add policies only on machine level.

So codegroups are resolved via union and policy levels via intersection, but there are two policy attributes "Exclusive" and "Level Final" building an exception to this. The following shows the attributes within the configuration console:



The Exclusive attribute applies when traversing the code group hierarchy within a single policy level. If the membership condition of a code group is matched and that code group is marked as Exclusive, then the permission set for that code group will be the only permission set granted to an assembly at this policy level. Even if the membership conditions of other code groups at the same policy level match assembly evidence, their permissions will not be added to the assembly's permission grant set.
You use the Exclusive attribute to force an assembly to have no more permissions than those specified by a specific permission set. For example, if you wanted to guarantee that a given assembly did not receive any permission, you would create a code group with a membership condition matching some evidence for that Assembly and give it the Nothing permission set. Remind that this attribute is valid only within one policy level.
Another policy statement attribute that alters default policy resolution is Level Final. If you place a Level Final on a code group at a given policy level and the assembly has evidence that matches the membership condition of that code group, none of the policy levels below that policy level will be evaluated. The hierarchy of policy levels starts with Enterprise at the top, then goes down to Machine, and finally ends at User.
The practical application of Level Final is to force a given policy on a system, preventing lower level policies from changing the policy in any way.

By default, the appdomain policy level is created with a single codegroup containing all code and granting full trust. Remember that the appdomain policy level will be combined with the enterprise, machine, and user policy levels in the most restrictive way. You cannot grant more rigths to an assembly through a custom appdomain policy level than could through any other policy level. It is therefore OK for the default appdomain policy to grant all rights. [9] [16]

2.6 Security Checks

There are several security checks that can be made against permissions. Those are coarse divided in demands, requests and overrides. Not all of the following security actions can be applied imperatively but all may be set the declarative way. Also be aware that not all of the actions can be applied to all types of targets.

2.6.1 Declarative vs. Imperative Security Checks

You can use two different kinds of syntax when coding permissions, declarative and imperative. The main difference between these two is that declarative calls are evaluated at compile time while imperative calls are evaluated at runtime. Note that compile time means during JIT compilation (IL to native).

Using imperative security, the user can not find out from your assembly itself what kind of permission is required or not wanted. But if you use declarative security, the request is contained in the assembly's metadata, which can be viewed by ildasm.exe. This can help administrators to setup security policy. See following example of a declarative check:

```
[FileIOPermissionAttribute(SecurityAction.Deny, ViewAndModify = @"D:\")]
public class TestClass
{
    public TestClass() {...}
    public void TestMethod() {...}
}
```

Using declarative attributes, in most cases, it is "staticly" decided whether an assembly is to be loaded or not, class to be created or not, and method to be called or not. While if using imperative command, all these decisions are made at run time. Your code can even say: "If the following condition is true than I do not want to do a stack walk…". So imperative commands are more flexible. [10]. See following example of an imperative check:

```
public class TestClass
{
    public TestClass() {}
    public void TestMethod()
    {
        FileIOPermission perm = new FileIOPermission(
            FileIOPermissionAccess.PathDiscovery | FileIOPermissionAccess.Read,
            @"D:\");
        perm.Deny();

        // do something
    }
}
```

Keep in mind that declarative CAS statements use attributes that invoke the permission classes. Some permission classes may not have corresponding attributes, and some attributes may not support all the security actions and/or all the attribute targets that you might expect.

2.6.2 Demands

Demands are used to ensure that every caller who calls your code (directly or indirectly) has been granted the demanded permission. This is accomplished by performing a stack walk. When demanded for a permission, the runtime's security system walks the call stack, comparing the granted permissions of each caller to the permission being demanded. If any caller in the call stack is found without the demanded

permission then a SecurityException is thrown. Different assemblies as well as different methods in the same assembly are checked by the stack walk. Be aware that demands are not used against assemblies. This action can only be put at class or method level. [4]

- Demand - have the CLR parse the whole call stack to check if all the assemblies involved in the call have the required permission. Forces a SecurityException at run time if all callers higher in the call stack have not been granted the permission specified by the current instance. Note that according to the documentation, most classes in .NET Framework already have demands associated with them. For example, take the StreamReader class. StreamReader automatically demands FileIOPermission. So placing another demand just before it causes an unnecessary stack walk.

- LinkDemand - A “LinkDemand” only checks the immediate caller (direct caller) of your code. That means it doesn't perform a stack walk. Linking occurs when your code is bound to a type reference, including function pointer references and method calls. A “LinkDemand” can only be applied declaratively because determinatin of whether the caller meets the demand conditions occur before the code is run.

- InheritanceDemand - Inheritance demands can be applied to classes or methods. If it is applied to a class, then all the classes that derive from this class must have the specified permission. If it is applied to a method, then all the classes that derive from this class must have the specified permission to override that method. An “InheritanceDemand” can only be applied declaratively because determination of whether the subclasses meet the demand conditions occur before the code is run.

2.6.3 Requests

Imagine the usecase of having a windows forms with a whole bunch of textboxes, checkboxes and radiobuttons the user needs to fill. After doing this he wants to apply his inputdata. Internally, the entered data wants to be stored in a file. Now this is the moment a security exception is throw because permission is set to allow FileIO. So all his entering the data was without a sense. For this case it would be preferable that the needed permission would be checked before data is entered.
Therefore requests can be used. They request permissions prior to loading an assembly, so an imperative use makes no sense. They can only be used the declarative way so checks can be made at compiletime. Be aware that requests are only valid against assemblies, not classes or methods. [4]

- RequestMinimum - You can use RequestMinimum to specify the permissions your code must have in order to run. The code will be only allowed to run if all the required permissions are granted by the security policy otherwise an exception would be thrown.

- RequestOptional -Using RequestOptional, you can specify the permissions your code can use, but not required in order to run. If somehow your code has not been granted the optional permissions, then you must handle any exceptions that are thrown while code segments that need these optional permissions are being executed. There are certain things to keep in mind when working with RequestOptional.

  If you use RequestOptional with RequestMinimum, no other permissions will be granted except these two, if allowed by the security policy. Even if the security policy allows additional permissions to your assembly, they won't be granted. Look at this code segment:

```
[assembly: FileIOPermission(SecurityAction.RequestMinimum, Read = "C:\\")]
[assembly: FileIOPermission(SecurityAction.RequestOptional, Write = "C:\\")]
```

  The only permissions that this assembly will have are read and write permissions to the file system. What if it needs to show a UI? Then the assembly still gets loaded but an exception will be thrown when the line that shows the UI is executing, because even though the security policy allows UIPermission, it is not granted to this assembly.

Note that, like RequestMinimum, RequestOptional doesn't prevent the assembly from being loaded, but throws an exception at run time if the optional permission has not been granted.

- RequestRefuse - You can use RequestRefuse to specify the permissions that you want to ensure will never be granted to your code, even if they are granted by the security policy. If your code only wants to read files, then refusing write permission would ensure that your code cannot be misused by a malicious attack or a bug to alter files.

The defaults for the various permission sets are “Nothing” (RequestMinimum), “FullTrust” (RequestOptional) and “Nothing” (RequestRefuse). Since the permission set restriction is calculated by taking the minimum request and unioning with the optional request and the default optional request is FullTrust, unless an Optional request is made, the union of the minimum request and the FullTrust request will result in a FullTrust set. So unless you specify a RequestOptional set, the permissions your application runs under will not be limited according to its default settings stated above. [4]

2.6.4 Overrides

Sometimes you need to override certain security checks. You can do this by altering the behavior of a permission stack walk using these three methods. [4]

- Assert - You can call the Assert method to stop the stack walk from going beyond the current stack frame. So the callers above the method that has used Assert are not checked. If you can trust the upstream callers, then using Assert would do no harm.

  RevertAssert is a static method of CodeAccessPermission. It is used after an Assert to cancel the Assert statement. So if the code below RevertAssert is accessing some protected resources, then a normal stack walk would be performed and all callers would be checked. If there's no Assert for the current stack frame, then RevertAssert has no effect. It is a good practice to place the RevertAssert in a finally block, so it will always get called.

  Note that if asserts are not handled carefully it may lead into luring attacks where malicious code can call your code through trusted code.

- Deny - Calling SecurityAction.Deny on the permission class object has the opposite effect of calling Assert: it blocks a stack walk on the method that called deny, even if the assembly had enough rights to have the security check succeed.

  RevertDeny is used to remove a previous Deny statement from the current stack frame.

- PermitOnly - Finally, calling SecurityAction.PermitOnly is similar to Deny: both cause stack walks to fail when they would otherwise succeed. The difference is that Deny specifies permissions that will cause the stack walk to fail, but PermitOnly specifies the only permissions that do not cause the stack walk to fail. You can use PermitOnly instead of Deny when it is more convenient to describe resources that can be accessed instead of resources that cannot be accessed.

Normally, a security check examines every caller in the call stack to ensure that each caller has been granted the specified permission. However, you can override the outcome of security checks by calling Assert, Deny, or PermitOnly on an individual permission object or a permission set object. Depending on which of these methods you call, you can cause the security check to succeed or fail, even though the permissions of all callers on the stack might not have been checked.

Every time one method calls another method, a new frame is generated on the call stack to store information about the method being called. (Using constructors and accessing properties are considered method calls in this context.) Each stack frame includes information about any calls the method makes to Assert, Deny, or PermitOnly. If a caller uses more than one Assert, Deny, or PermitOnly in the same method call, the runtime applies the following processing rules, which can affect override behaviors:

- If, during the stack walk, the runtime discovers more than one override of the same type (that is, two calls to Assert) in one stack frame, the second override causes an exception to be thrown.

- When different overrides are present in the same stack frame, the runtime processes these overrides in the following order: PermitOnly, then Deny, and finally Assert.

To replace an override, first call the appropriate revert method (for example, RevertAssert) and then apply the new override.

Make sure that stack-walk overrides should never be made in a class constructor because class constructor code is not guaranteed to execute at any particular point or in any particular context. Because the state of the call stack in a class constructor is not well defined, stack walk overrides placed in constructors can produce unexpected and undesired results.

Application developers do not usually need to use Assert, Deny, or PermitOnly, and component and class library developers rarely need to use them. However, security overrides are appropriate in some situations, which are described in the Assert, Deny, and PermitOnly topics.

Note that if you perform an override (Deny, Assert, or PermitOnly), you must revert the permission before you can perform the same kind of override in the same stack frame (that is, method). Otherwise, a SecurityException is thrown. For example, if you deny a permission, P, you must revert that permission before you can deny another permission, Q, in the same method.

Use one of the static methods listed in the following table to revert an override.

- CodeAccessPermission.RevertAll - Causes all previous overrides for the current frame to be removed and no longer in effect.
- CodeAccessPermission.RevertAssert - Causes any previous Assert for the current frame to be removed and no longer in effect.
- CodeAccessPermission.RevertDeny - Causes any previous Deny for the current frame to be removed and no longer in effect.
- CodeAccessPermission.RevertPermitOnly - Causes any previous PermitOnly for the current frame to be removed and no longer in effect.

3 Overview on Partial Trust

Assemblies that are not granted full trust are called partial trusted. Those assemblies are not allowed to call fully trusted ones, unless the latter contain the AllowPartiallyTrustedCallersAttribute (APTCA) attribute:

```
[assembly: AllowPartiallyTrustedCallersAttribute()]
```

This is a risk mitigation strategy designed to ensure your code cannot inadvertently be exposed to partial trust (potentially malicious) code. The common language runtime silently adds a link demand for the FullTrust permission set to all publicly accessible members on types in a strong named assembly. If you include APTCA, you suppress this link demand.
If you use APTCA, your code is immediately more vulnerable to attack and, as a result, it is particularly important to review your code for security vulnerabilities. Use APTCA only where it is strictly necessary.

The following code demonstrates what can be done, if code is blindly running with full trust:

```
class TestClass
{
    private void TestMethod(string text)
    {
        Console.WriteLine(text);
    }
}

class Caller
{
    static void Main()
    {
        TestClass instance = new TestClass();
        Type t = typeof(TestClass);
        MethodInfo mi = t.GetMethod(
            "TestMethod",
            BindingFlags.Instance | BindingFlags.NonPublic);
        mi.Invoke(instance, new Object[] { "hello" });

        Console.ReadLine();
    }
}
```

Here full trust allows the code using reflection to access a private member of a foreign class. This could be used to try calling exposed methods with different or even senseless parameters. Maybe it discovers a security lack because of a missing parameter input validation, just because the provided class is set to full trust.

In the context of server-side Web applications, use APTCA whenever your assembly needs to support partial trust callers. This situation can occur in the following circumstances:

- Your assembly is to be called by a partial trust Web application. These are applications for which the <trust> level is set to something other than Full. For more information about partial trust Web applications and using APTCA in this situation, see [12].
- Your assembly is to be called by another assembly that has been granted limited permissions by the code access security administrator.
- Your assembly is to be called by another assembly that refuses specific permissions by using SecurityAction.RequestRefuse or SecurityAction.RequestOptional. These make the calling assembly a partial trust assembly.

- Your assembly is to be called by another assembly that uses a stack walk modifier (such as Deny or PermitOnly) to constrain downstream code.

So always be aware that an application with Full Trust permissions has unrestricted access to resources such as the file system and the registry, potentially allowing your application (and the end user's system) to be exploited by malicious code.

4 List of Predefined Permissions and its Parameters

Information about .NET security permissions is spread on MSDN and several weblogs throughout the community. The intend of this list is to provide a central place to find information on permission, their parameters and properties.

For getting an overview on the large list of permission, we divide them to those which can be modified via the “machine-level” hive in runtime security policy settings of the .NET framework 2.0 Configuration Management Console snap-in and those which are not.
Instances of permission classes can be manipulated by properties and parameters given in its constructors or methods. Some permissions map parameters to properties and vice versa. The following document does not list all those methods explicitly. Here we lay emphasis on the parameters and so the properties only.

PermissionState

By default each permission for a resource contains a parameter of type PermissionState. It is an enumerator specifying whether a permission should have all or no access to resources at creation.

- None - No access to the resource protected by the permission.
- Unrestricted - Full access to the resource protected by the permission.

This default permission is not explicitly listed in the list of all permissions. E.g. System.Net.DnsPermission does not contain any other parameters.

4.1 Permissions represented via Configuration Console

4.1.1 System.Data.SqlClient.SqlClientPermission

Enables the .NET Framework Data Provider for SQL Server to help ensure that a user has a security level adequate to access a data source. The class inherits from ResourcePermissionBase which in turn is inherited from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. The corresponding entry in configuration console is called “SQL Client”. Class is contained in module “System.Data.dll”.

Parameters and Properties:

AllowBlankPassword : Bool
Gets a value indicating whether a blank password is allowed. This parameter belongs to an obsolete constructor “SqlClientPermission(PermissionState state, bool allowBlankPassword) “ which should not be used any more from .NET framework 2.0 on.

ConnectionString : String
A permitted connection string.

Restrictions : String
String that identifies connection string parameters that are allowed or disallowed.

Behavior : System.Data.KeyRestrictionBehavior
Defines access levels and can be set to AllowOnly and PreventUsage:

- AllowOnly - Default. Identifies the only additional connection string parameters that are allowed.
- PreventUsage - Identifies additional connection string parameters that are not allowed.

Note:

When more than one rule is specified, the more restrictive is selected.

When using code access security permissions for ADO.NET, the correct pattern is to start with the most restrictive case (no permissions at all) and then add the specific permissions that are needed for the particular task that the code needs to perform. The opposite pattern, starting with all permissions and then denying a specific permission, is not secure, because there are many ways of expressing the same connection string. For example, if you start with all permissions and then attempt to deny the use of the connection string "server=someserver", the string "server=someserver.mycompany.com" would still be allowed. Always starting by granting no permissions at all, you reduce the chances that there are holes in the permission set.

The IsUnrestricted property from baseclass DBDataPermission takes precedence over the AllowBlankPassword property. Therefore, if you set AllowBlankPassword to false, you must also set IsUnrestricted to false to prevent a user from making a connection using a blank password.

4.1.2 System.Diagnostics.EventLogPermission

Allows to control accessing the eventlog. The class inherits from ResourcePermissionBase which in turn is inherited from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. The corresponding entry in configuration console is called “Event Log”. Class is contained in module “System.dll”.

Parameters and Properties:

MachineName : String
The name of the computer on which to read or write events.

PermissionAccess : System.Diagnostics.EventLogPermissionAccess
Defines access levels used by EventLog permission classes and can be set to Administer, Audit, Browse, Instrument, None, Write:

- Administer – The EventLog can create an event source, read existing logs, delete event sources or logs, respond to entries, clear an event log, listen to events, and access a collection of all event logs.
- Audit – The EventLog can read existing logs, delete event sources or logs, respond to entries, clear an event log, listen to events, and access a collection of all event logs. Note: This member is now obsolete, use Administer instead.
- Browse – The EventLog can read existing logs. Note: This member is now obsolete, use Administer instead.
- Instrument – The EventLog can read or write to existing logs, and create event sources and logs. Note: This member is now obsolete, use Write instead.
- None – The EventLog has no permissions.
- Write – The EventLog can write to existing logs, and create event sources and logs.

4.1.3 System.Diagnostics.PerformanceCounterPermission

Allows to control accessing the performance counter. The class inherits from ResourcePermissionBase which is in turn inherited from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. The corresponding entry in configuration console is called “Performance Counter”. Class is contained in module “System.dll”.

Defines access levels, the name of the computer to use, and the category associated with the performance counter.

Parameters and Properties:

CategoryName : string
The name of the performance counter category (performance object) with which the performance counter is associated.

MachineName: string
The server on which the performance counter and its associate category reside.

PermissionAccess : System.Diagnostics.PerformanceCounterPermissionAccess
Defines access levels and can be set to Administer, Browse, Instrument, None, Read and Write:

- Administer – The PerformanceCounter can read, write, and create categories.
- Browse – The PerformanceCounter can read categories.
- Instrument – The PerformanceCounter can read and write categories.
- None – The PerformanceCounter has no permissions.
- Read – The PerformanceCounter can read categories.
- Write – The PerformanceCounter can write categories.

4.1.4 System.DirectoryServices.DirectoryServicesPermission

Allows to control accessing Active Directory. The class inherits from ResourcePermissionBase which in turn is inherited from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. The corresponding entry in configuration console is called “Directory Services”. Class is contained in module “System. DirectoryServices.dll”.

Parameters and Properties:

Path : String
The path of the Active Directory node to which the permissions apply.

PermissionAccess : System.DirectoryServices.DirectoryServicesPermissionAccess
Defines the type of access you want to allow. Can be set to Browse, None and Write:

- Browse – Reading the Active Directory tree is allowed.
- None – There are no permissions.
- Write – Reading, writing, deleting, changing and adding to the Active Directory tree are allowed.

4.1.5 System.Drawing.Printing.PrintingPermission

Allows to control accessing the printer. The class inherits directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. The corresponding entry in configuration console is called “Printing”. Class is contained in module “System.Drawing.dll”.

Parameters and Properties:

PrintingLevel : System.Drawing.Printing.PrintingPermissionLevel

Defines the type of access you want to allow. Can be set to Allprinting, DefaultPrinting, NoPrinting and SafePrinting:

- AllPrinting – Provides full access to all printers.
- DefaultPrinting – Provides printing programmatically to the default printer, along with safe printing through showing a less restricted dialog box. DefaultPrinting is a subset of AllPrinting.
- NoPrinting – Prevents access to printers. NoPrinting is a subset of SafePrinting.
- SafePrinting - Provides printing only from showing a restricted dialog box. SafePrinting is a subset of DefaultPrinting.

4.1.6 System.Messaging.MessageQueuePermission

Allows to control accessing message queues. The class inherits directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. The corresponding entry in configuration console is called “Message Queue”. Class is contained in module “System.Messaging.dll”.

A message queue can be described in two ways. Either by defining the machinename where the message queue is located, a label used to describe the queue, a category to group queues logically and the type of access you want to allow (MessageQueuePermissionAccess). Or by the path where it is located and again the MessageQueuePermissionAccess.

Parameters and Properties:

Category : String
The queue category (Message Queuing type identifier).

Label : String
The queue description.

MachineName : String
The name of the computer where the Message Queuing queue is located.

PermissionAccess : MessageQueuePermissionAccess
Defines the type of access you want to allow. Can be set to Administer, Browse, None, Peek, Receive and Send.

- Administer - The MessageQueue can look at the queues that are available, read the messages in the queue, send and receive messages.
- Browse – The MessageQueue can look at the queues that are available.
- None – The MessageQueue has no permissions.
- Peek – The MessageQueue can look at the queues that are available and read the messages in the queue.
- Receive – The MessageQueue can look at the queues that are available, read the messages in the queue and receive messages.
- Send – The MessageQueue can look at the queues that are available and send messages.

4.1.7 System.Net.DnsPermission

Allows to control accessing DNS servers on the network. The class inherits directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. The corresponding entry in configuration console is called “DNS”. Class is contained in module “System.dll”.

Parameters and Properties: only default parameters (PermissionState)

4.1.8 System.Net.SocketPermission

Allows to control making or accepting connections on a transport address. The class inherits directly from CodeAccessPermission and is contained in .NET framework since version 1.0. The corresponding entry in configuration console is called “Socket Access”. Class is contained in module “System.dll”.

Defines the type of networkaccess you want to allow. You also must specify the targethost, its portnumber and the used transporttype.

Parameters and Properties:

Access : System.Net.NetworkAccess
Specifies network access permissions. It is used with the WebPermission and SocketPermission classes. Can be set to Accept or Connect.

- Accept - Indicates that the application is allowed to accept connections from the Internet on a local resource. Notice that this is a protection for the local host that uses Accept to grant access to a local resource (address/port). At the time a socket tries to bind to this local resource a permission check is performed to see if an Accept exists on that resource.
- Connect - Indicates that the application is allowed to connect to specific Internet resources. Notice that, in the case of remote host resource, no check is performed to see that Connect permissions exist. This is because the port of a connecting remote host is unknown and not suitable permissions can be built in advance. It is the application responsibility to check the permissions of the remote host trying to connect to a listening socket.

HostName : String
The host name for the transport address.

PortNumber : String
The port number for the transport address.

Transport : System.Net.TransportType
Defines transport types and can be set to All, ConnectionLess, ConnectionOriented, Tcp or Udp.

- All – All transport types.
- ConnectionLess - The transport type is connectionless, such as UDP. Specifying this value has the same effect as specifying Udp.
- ConnectionOriented - The transport is connection oriented, such as TCP. Specifying this value has the same effect as specifying Tcp.
- Tcp – TCP transport.
- Udp – UDP transport.

4.1.9 System.Net.WebPermission

Allows to control accessing HTTP Internet resources. The class inherits directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. The corresponding entry in configuration console is called “Web Access”. Class is contained in module “System.dll”.

WebPermission provides a set of methods and properties to control access to Internet resources. Create a WebPermission instance by calling its constructor using one of the following sets of parameters:

- No parameters. The default PermissionState is None.
- A PermissionState. Specify either Unrestricted to allow any URI to be used in the target class, or None to allow access only to URIs that you specify through the use of the AddPermission method.
- A NetworkAccess value and a URI string. The specified URI has permissions granted by the NetworkAccess value.
- A NetworkAccess specifier and URI regular expression.

The ConnectList and AcceptList hold the URIs to which you have granted access permission. To add a URI to either of these lists, use AddPermission. If you pass Accept as the NetworkAccess parameter, the URI will be added to the AcceptList. WebPermission will allow connections to your target class with URIs matching the AcceptList.

Parameters and Properties:

Access : System.Net.NetworkAccess
Specifies network access permissions. It is used with the WebPermission and SocketPermission classes. Can be set to Accept or Connect.

- Accept - Indicates that the application is allowed to accept connections from the Internet on a local resource. Notice that this is a protection for the local host that uses Accept to grant access to a local resource (address/port). At the time a socket tries to bind to this local resource a permission check is performed to see if an Accept exists on that resource.

- Connect - Indicates that the application is allowed to connect to specific Internet resources. Notice that, in the case of remote host resource, no check is performed to see that Connect permissions exist. This is because the port of a connecting remote host is unknown and not suitable permissions can be built in advance. It is the application responsibility to check the permissions of the remote host trying to connect to a listening socket.

UriString : string
The URL you want the permission to be valid for.

UriRegEx : Regex
A regular expression describing URLs you want the permission to be valid for.

Notes:

To Deny access to an Internet resource, you must Deny access to all the possible paths to that resource. A better approach is to allow access to the specific resource only.

You need to Deny access using only the resource canonical path. There is no need to use all the path's syntactical variations.

User name and default port information is stripped from the Uri before the comparison with the regular expression argument that is supplied to the WebPermission(NetworkAccess,Regex) constructor. If the regular expression contains user information or the default port number, then all incoming Uris will fail to match the regular expression.

4.1.10 System.Security.Permissions.EnvironmentPermission

Allows to control accessing system and user environment variables and its values. The class inherits directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. The corresponding entry in configuration console is called “Environment Variables”. Class is contained in module “mscorlib.dll”.

Environment variable names are designated by one or more case-insensitive name lists separated by semicolons, with separate lists for read and write access to the named variables. Write access includes the ability to create and delete environment variables as well as to change existing values.

Parameters and Properties:

PathList : String
A semicolon-separated list of environment variables to which access is granted.

Flag : System.Security.Permissions.EnvironmentPermissionAccess
This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values. Specifies access to environment variables and can be set to AllAccess, NoAccess, Read and Write:

- AllAccess – Read and Write access to environment variables. AllAccess represents multiple EnvironmentPermissionAccess values and causes an ArgumentException when used as the flag parameter for the GetPathList method, which expects a single value.
- NoAccess – No access to environment variables. NoAccess represents no valid EnvironmentPermissionAccess values and causes an ArgumentException when used as the parameter for GetPathList, which expects a single value.
- Read – Only read access to environment variables is specified. Changing, deleting and creating environment variables is not included in this access level
- Write – Only write access to environment variables is specified. Write access includes creating and deleting environment variables as well as changing existing values. Reading environment variables is not included in this access level.

Notes:

Although NoAccess and AllAccess appear in EnvironmentPermissionAccess, they are not valid for use as the parameter for GetPathList because they describe no environment variable access types or all environment variable access types, respectively, and GetPathList expects a single environment variable access type.

EnvironmentPermission grants permission for access to the environment variable and its value. To deny access to a variable and its value, you must deny access to it and any other variable which contains the same value. For example, to Deny access to the "TMP" variable and its value, "%USERPROFILE%\Local Settings\Temp", you must Deny access to "TMP", "TEMP" and any other variable that you can use to access that value. A better technique to deal with multiple paths is to use a combination of PermitOnly and Deny. For more information on this subject and the use of PermitOnly with Deny, see Canonicalization Problems Using Deny.

4.1.11 System.Security.Permissions.FileDialogPermission

Allows to control accessing files and folders through a file dialog. The class inherits directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0 and it cannot be inherited.The corresponding entry in configuration console is called "File Dialog". Class is contained in module "mscorlib.dll".

Parameters and Properties:

Access : System.Security.Permissions.FileDialogPermissionAccess
This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values. Specifies the kind of possible fileaccess for filedialogs. It can be set to None, Open, OpenSave and Save:

- None – No access to files.
- Open – Possibility to open files.
- OpenSave – Possibility to open and save files
- Save – Possibility to save files.

4.1.12 System.Security.Permissions.FileIOPermission

Allows to control accessing files and folders. The class inherits directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0.The corresponding entry in configuration console is called "File IO". Class is contained in module "mscorlib.dll".

Describes protected operations on files and folders. The File class helps provide secure access to files and folders. The security access check is performed when the handle to the file is created. By doing the check at creation time, the performance impact of the security check is minimized. Opening a file happens once, while reading and writing can happen multiple times. Once the file is opened, no further checks are done. If the object is passed to an untrusted caller, it can be misused. For example, file handles should not be stored in public global statics where code with less permission can access them.

Parameters and Properties:

Path : String
The absolute path of the file or directory.

Control : System.Security.AccessControl.AccessControlActions
Specifies the actions that are permitted for securable objects. This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values. This enumeration is new in the .NET Framework version 2.0. It can be set to Change, None and View:

- Change – Specifies write-only access.
- None – Specifies no access.
- View – Specifies read-only access.

Access : System.Security.Permissions.FileIOPermissionAccess
Defines actions that can be performed on the file(s) or folder(s) contained in the added path or pathlist. Access to a folder implies access to all the files it contains, as well as access to all the files and folders in its subfolders. For example, Read access to “C:\folder1\” implies Read access to “C:\folder1\file1.txt”, “C:\folder1\folder2\” and “C:\folder1\folder2\file2.txt”.
All these permissions are independent, meaning that rights to one do not imply rights to another. For example, Write permission does not imply permission to Read or Append. If more than one permission is desired, they can be combined using a bitwise OR. File permission is defined in terms of canonical absolute paths; calls should always be made with canonical file paths.
Can be set to AllAccess, Append, NoAccess, PathDiscovery, Read and Write:

- AllAccess – Append, Read, Write, and PathDiscovery access to a file or directory. AllAccess represents multiple FileIOPermissionAccess values and causes an ArgumentException when used as the access parameter for the GetPathList method, which expects a single value.
- Append – Access to append material to a file or directory. Append access includes the ability to create a new file or directory. To create files, code must also be granted both Append and either Write or Read access.
- NoAccess – No access to a file or directory. NoAccess represents no valid FileIOPermissionAccess values and causes an ArgumentException when used as the parameter for GetPathList, which expects a single value.
- PathDiscovery – Access to the information in the path itself. This helps protect sensitive information in the path, such as user names, as well as information about the directory structure revealed in the path. This value does not grant access to files or folders represented by the path.
- Read – Read access to the contents of the file or access to information about the file, such as its length or last modification time.
- Write – Write access to the contents of the file or access to change information about the file, such as its name. Also allows for deletion and overwriting.

AllFiles : System.Security.Permissions.FileIOPermissionAccess
Together with “AllLocalFiles” this parameter gives the opportunity to subdivide access control on files in pathlist. It now can be divided in all files and only files wich are local. Here it gets or sets the permitted access to all files in the added path or pathlist.
See “Access”-parameter above for values. This property gets or sets the permitted access to all files on the local computer and network drives. An individual FileIOPermissionAccess value can be checked for using a bitwise AND operation.

AllLocalFiles : System.Security.Permissions.FileIOPermissionAccess
Together with “AllLocalFiles” this parameter gives the opportunity to subdivide access control on files in pathlist. It now can be divided in all files and only files wich are local. Here it gets or sets the permitted access to all local files. See “Access”-parameter above for values. Local files are files contained on the local computer. Any files not accessed through a network drive are local files. An individual FileIOPermissionAccess value can be checked for using a bitwise AND operation.

Notes:

Unrestricted FileIOPermission grants permission for all paths within a file system, including multiple pathnames that can be used to access a single given file. To Deny access to a file, you must Deny all possible paths to the file. For example, if “\\server\share” is mapped to the network drive X, to Deny access to “\\server\share\file”, you must Deny “\\server\share\file”, “X:\file” and any other path that you can use to access the file. A better technique to deal with multiple paths is to use a combination of PermitOnly and Deny. In the above example you can PermitOnly “\\server\share”, then Deny “\\server\share\file”, eliminating alternate paths completely. For more information on this subject and the use of PermitOnly with Deny, see "Canonicalization Problems Using Deny" in Using the Deny Method.

Deny is most effective when used with the Windows NTFS file system. NTFS offers substantially more security than FAT32.

For performance reasons, PathDiscovery should only be granted to directories, not to files. For example, PathDiscovery permission should be granted to paths such as “C:\test” and “C:\test\”, not “C:\test\example.txt”.

To create files, code must also be granted both Append and either Write or Read access.

Although NoAccess and AllAccess appear in FileIOPermissionAccess, they are not valid for use as the parameter for GetPathList because they describe no file access types or all file access types, respectively, and GetPathList expects a single file access type.

4.1.13 System.Security.Permissions.IsolatedStorageFilePermission

Represents access to generic isolated storage capabilities. The class is inherited directly from CodeAccessPermission. It is contained in .NET framework since version 1.0. Class is contained in module “mscorlib.dll”. It builds the baseclass for “System.Security.Permissions.IsolatedStorageFilePermission”.

Isolated storage uses evidence to determine a unique storage area for use by an application or component. The identity of an assembly uniquely determines the root of a virtual file system for use by that assembly. Thus, rather than many applications and components sharing a common resource such as the file system or registry, each has its own file area inherently assigned to it.

Four basic isolation scopes are used when assigning isolated storage:

- User - Code is always scoped according to the current user. The same assembly will receive different stores when being run by different users.
- Machine - Code is always scoped according to the machine. The same assembly will receive the same stores when being run by different users on the same machine.
- Assembly - Code is identified cryptographically by strong name (for example, Microsoft.Office.* or Microsoft.Office.Word), by publisher (based on public key), by URL (for example, http://www.fourthcoffee.com/process/grind.htm), by site, or by zone.
- Domain - Code is identified based on evidence associated with the application domain. Web application identity is derived from the site's URL, or by the Web page's URL, site, or zone. Local code identity is based on the application directory path.

For definitions of URL, site, and zone, see UrlIdentityPermission, SiteIdentityPermission, and ZoneIdentityPermission.

These identities are grouped together, in which case the identities are applied one after another until the desired isolated storage is created. The valid groupings are User+Assembly and User+Assembly+Domain. This grouping of identities is useful in many different applications.

If data is stored by domain, user, and assembly, the data is private in that only code in that assembly can access the data. The data store is also isolated by the application in which it runs, so that the assembly does not represent a potential leak by exposing data to other applications.

Isolation by assembly and user could be used for user data that applies across multiple applications; for example, license information, or a user's personal information (name, authentication credentials, and so on) that is independent of an application.

IsolatedStorageContainment exposes flags that determine whether an application is allowed to use isolated storage and, if so, which identity combinations are allowed to use it. It also determines whether an application is allowed to store information in a location that can roam with a user (Windows Roaming User Profiles or Folder Redirection must be configured).

Parameters and Properties:

UserQuota : System.Int64
Gets or sets the quota on the overall size of each user's total store. Size in in bytes.

UsageAllowed : System.Security.Permissions.IsolatedStorageContainment
Gets or sets the type of isolated storage containment allowed. Can be set to the following values:

- AdministerIsolatedStorageByUser – Unlimited administration ability for the user store. Allows browsing and deletion of the entire user store, but not read access other than the user's own domain/assembly identity.
- ApplicationIsolationByMachine – Storage is isolated first by computer and then by application. This provides a data store for the application that is accessible in any domain context. The per-application data compartment requires additional trust because it potentially provides a "tunnel" between applications that could compromise the data isolation of applications in particular Web sites.
- ApplicationIsolationByRoamingUser – Storage is isolated first by user and then by application evidence. Storage will roam if Windows user data roaming is enabled. This provides a data store for the application that is accessible in any domain context. The per-application data compartment requires additional trust because it potentially provides a "tunnel" between applications that could compromise the data isolation of applications in particular Web sites.
- ApplicationIsolationByUser – Storage is isolated first by user and then by application. Storage is also isolated by computer. This provides a data store for the application that is accessible in any domain context. The per-application data compartment requires additional trust because it potentially provides a "tunnel" between applications that could compromise the data isolation of applications in particular Web sites.
- AssemblyIsolationByMachine – Storage is isolated first by computer and then by code assembly. This provides a data store for the assembly that is accessible in any domain context. The per-assembly data compartment requires additional trust because it potentially provides a "tunnel" between applications that could compromise the data isolation of applications in particular Web sites.
- AssemblyIsolationByRoamingUser – Storage is isolated first by user and then by assembly evidence. Storage will roam if Windows user data roaming is enabled. This provides a data store for the assembly that is accessible in any domain context. The per-assembly data compartment requires additional trust because it potentially provides a "tunnel" between applications that could compromise the data isolation of applications in particular Web sites.
- AssemblyIsolationByUser – Storage is isolated first by user and then by code assembly. Storage is also isolated by computer. This provides a data store for the assembly that is accessible in any domain context. The per-assembly data compartment requires additional trust because it potentially provides a "tunnel" between applications that could compromise the data isolation of applications in particular Web sites.
- DomainIsolationByMachine – Storage is isolated first by computer and then by domain and assembly. Data can only be accessed within the context of the same application and only when run on the same computer. This is helpful when a third-party assembly wants to keep a private data store.
- DomainIsolationByRoamingUser – Storage is isolated first by user and then by domain and assembly. Storage will roam if Windows user data roaming is enabled. Data can only be accessed within the context of the same application and only when run by the same user. This is helpful when a third-party assembly wants to keep a private data store.
- DomainIsolationByUser – Storage is isolated first by user and then by domain and assembly. Storage is also isolated by computer. Data can only be accessed within the context of the same application and only when run by the same user. This is helpful when a third-party assembly wants to keep a private data store.
- None – Use of isolated storage is not allowed.
- UnrestrictedIsolatedStorage – Use of isolated storage is allowed without restriction. Code has full access to any part of the user store, regardless of the identity of the domain or assembly. This use of isolated storage includes the ability to enumerate the contents of the isolated storage data store.

4.1.14 System.Security.Permissions.ReflectionPermission

Controls access to metadata through the System.Reflection APIs. The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0.The corresponding entry in configuration console is called “Reflection”. Class is contained in module “mscorlib.dll”.

Parameters and Properties:

Flag : Type:System.Security.Permissions.ReflectionPermissionFlag
Without ReflectionPermission, code can only access the public members of loaded assemblies. This includes, but is not limited to, unrestricted access to Object.GetType, access to public exported types through Type.GetType, and access to GetTypeFromHandle. Some properties of Type, such as FullName and Attributes, are accessible without ReflectionPermission.
Code with the appropriate ReflectionPermission has access to the public, protected, and even private members of any loaded Type. This includes access to Module, Assembly, BaseType, and GetInterfaces.
Can be set to AllFlags, MemberAccess, NoFlags, ReflectionEmit and TypeInformation:

- AllFlags – TypeInformation, MemberAccess, and ReflectionEmit are set.
- MemberAccess – Invocation of operations on all type members is allowed. If this flag is not set, only invocation of operations on visible type members is allowed.
- NoFlags – No reflection is allowed on types that are not visible.
- ReflectionEmit – Use of System.Reflection.Emit is allowed.
- TypeInformation – Reflection is allowed on members of a type that is not visible.

Notes:

Because ReflectionPermission can provide access to private class members and metadata, it is recommended that ReflectionPermission not be granted to Internet code.

4.1.15 System.Security.Permissions.RegistryPermission

Controls the ability to access registry variables. The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. The corresponding entry in configuration console is called “Registry”. Class is contained in module “mscorlib.dll”.

Parameters and Properties:

PathList : String
A list of semicolon-separated registry variables to which access is granted.

Control : System.Security.AccessControl.AccessControlActions
Specifies the actions that are permitted for securable objects. This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values. This enumeration is new in the .NET Framework version 2.0. It can be set to Change, None and View:

- Change – Specifies write-only access.
- None – Specifies no access.
- View – Specifies read-only access.

Access : System.Security.Permissions.RegistryPermissionAccess
Specifies the permitted access to registry keys and values. This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values. RegistryPermissionAccess values are independent; rights to one type of access do not imply rights to another. For instance, Write permission does not imply permission to Read or Create.
Although NoAccess and AllAccess appear in RegistryPermissionAccess, they are not valid for use as the parameter for GetPathList because they describe no registry variable access types or all registry variable access types, respectively, and GetPathList expects a single registry variable access type.
Can be AllAccess, Create, NoAccess, Read and Write:

- AllAcces – Create, Read, and Write access to registry variables. AllAccess represents multiple RegistryPermissionAccess values and causes an ArgumentException when used as the access parameter for the GetPathList method, which expects a single value.
- Create – Create access to registry variables.
- NoAccess – No access to registry variables. NoAccess represents no valid RegistryPermissionAccess values and causes an ArgumentException when used as the parameter for GetPathList, which expects a single value.
- Read – Read access to registry variables.

- Write – Write access to registry variables.

Notes:

RegistryPermission describes protected operations on registry variables. Registry variables should not be stored in memory locations where code without RegistryPermission can access them. If the registry object is passed to an untrusted caller it can be misused.

Registry permission is defined in terms of canonical absolute paths; checks should always be made with canonical pathnames. Key access implies access to all values it contains and all variables under it.

RegistryPermission grants permission for all paths to a key, including both HKEY_CURRENT_USER and HKEY_USERS. To Deny access to a key, you must Deny all possible paths to the key. For example, to Deny access to “HKEY_CURRENT_USER\Software\Microsoft\Cryptography”, you must Deny “HKEY_CURRENT_USER\Software\Microsoft\Cryptography”, “HKEY_USERS\.......\Software\Microsoft\Cryptography” and any other path that you can use to access the key. A better technique to deal with multiple paths is to use a combination of PermitOnly and Deny. For more information on this subject and the use of PermitOnly with Deny, see Canonicalization Problems Using Deny.

4.1.16 System.Security.Permissions.SecurityPermission

Describes a set of security permissions applied to code. The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. The corresponding entry in configuration console is called “Security”. Class is contained in module “mscorlib.dll”.

Parameters and Properties:

Flag : System.Security.Permissions.SecurityPermissionFlag
Specifies access flags for the security permission object. This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values. Can be set to AllFlags, Assertion, BindingRedirects, ControlAppDomain, ControlDomainPolicy, ControlEvidence, ControlPolicy, ControlPrincipal, ControlThread and Execution:

- AllFlags – The unrestricted state of the permission.
- Assertion – Ability to assert that all this code's callers have the requisite permission for the operation.
- BindingRedirects – Permission to perform explicit binding redirection in the application configuration file. This includes redirection of .NET Framework assemblies that have been unified as well as other assemblies found outside the .NET Framework.
- ControlAppDomain – Ability to create and manipulate an AppDomain.
- ControlDomainPolicy – Ability to specify domain policy.
- ControlEvidence – Ability to provide evidence, including the ability to alter the evidence provided by the common language runtime. This is a powerful permission that should only be granted to highly trusted code.
- ControlPolicy – Ability to view and modify policy. This is a powerful permission that should only be granted to highly trusted code.
- ControlPrincipal – Ability to manipulate the principal object.
- ControlThread – Ability to use certain advanced operations on threads.
- Execution – Permission for the code to run. Without this permission, managed code will not be executed. But functionality seen to be harmful is not permitted to be used. E.g. creating and managing own application domains is forbidden.
  This flag has no effect when used dynamically with stack modifiers such as Deny, Assert, and PermitOnly.
- Infrastructure – Permission to plug code into the common language runtime infrastructure, such as adding Remoting Context Sinks, Envoy Sinks and Dynamic Sinks.
- NoFlags – No security access.
- RemotingConfiguration – Permission to configure Remoting types and channels.
- SerializationFormatter – Ability to provide serialization services. Used by serialization formatters.

- SkipVerification – Ability to skip verification of code in this assembly. Code that is unverifiable can be run if this permission is granted. This is a powerful permission that should be granted only to highly trusted code. This flag has no effect when used dynamically with stack modifiers such as Deny, Assert, and PermitOnly.
- UnmanagedCode – Ability to call unmanaged code. Since unmanaged code potentially allows other permissions to be bypassed, this is a dangerous permission that should only be granted to highly trusted code. It is used for such applications as calling native code using PInvoke or using COM interop.

Notes:

Many of these flags are powerful and should only be granted to highly trusted code.

4.1.17 System.Security.Permissions.StorePermission

Allows to control accessing stores containing X.509 certificates. The class inherits directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 2.0. The corresponding entry in configuration console is called “X509 Store”. Class is contained in module “System.dll”.

Parameters and Properties:

Flag : System.Security.Permissions.StorePermissionFlags
Defines the type of access you want to allow. Can be set to AddToStore, AllFlags, CreateStore, DeleteStore, EnumerateCertificates, EnumerateStores, NoFlags, OpenStore, RemoveFromStore:

- AddToStore - The ability to add a certificate to a store. For security reasons, this ability should be granted only to highly trusted code.
- AllFlags - The ability to perform all certificate and store operations.
- CreateStore - The ability to create a new store.
- DeleteStore - The ability to delete a store.
- EnumerateCertificates - The ability to enumerate the certificates in a store. For privacy reasons, this ability should be granted only to fully trusted code.
- EnumerateStores - The ability to enumerate the stores on a computer.
- NoFlags - Permission is not given to perform any certificate or store operations.
- OpenStore - The ability to open a store. The ability to open a store does not include the ability to enumerate certificates (which raises privacy concerns) or to add or remove certificates (which raises security concerns).
- RemoveFromStore - The ability to remove a certificate from a store. This ability should be granted only to highly trusted code because removing a certificate can result in a denial of service.

4.1.18 System.Security.Permissions.UIPermission

Controls the permissions related to user interfaces and the clipboard. The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. The corresponding entry in configuration console is called “User Interface”. Class is contained in module “mscorlib.dll”. Drawing and user input events in windows are user interfaces.

Parameters and Properties:

WindowFlag : System.Security.Permissions.UIPermissionWindow
Specifies the type of windows that code is allowed to use. Can be set to AllWindows, NoWindows, SafeSubWindows and SafeTopLevelWindows:

- AllWindows – Users can use all windows and user input events without restriction.
- NoWindows – Users cannot use any windows or user interface events. No user interface can be used.

- SafeSubWindows – Users can only use SafeSubWindows for drawing, and can only use user input events for user interface within that subwindow. Examples of SafeSubWindows are a MessageBox, common dialog controls, and a control displayed within a browser.
- SafeTopLevelWindows – A toplevel window is the mainwindow of an application and is used as parent for further windows like forms, message boxes etc. Safe means that it is permitted to manipulate window properties like size, visibility, transparency, title or the location on the desktop. Those are then readonly. These restrictions help prevent potentially harmful code from attacking via hidden windows or window manipulation like faked login dialogs.
  Users can only use SafeTopLevelWindows and SafeSubWindows for drawing, and can only use user input events for the user interface within those top-level windows and subwindows. These special windows, for use by partially-trusted code, are guaranteed to be clearly labeled and have minimum and maximum size restrictions.

ClipboardFlag : System.Security.Permissions.UIPermissionClipboard
Specifies the type of clipboard access that is allowed to the calling code. Can be set to AllClipboard, NoClipboard and OwnClipboard:

- AllClipboard – Clipboard can be used without restriction.
- NoClipboard – Clipboard cannot be used.
- OwnClipboard – The ability to put data on the clipboard (Copy, Cut) is unrestricted. Intrinsic controls that accept Paste, such as text box, can accept the clipboard data, but user controls that must programmatically read the clipboard cannot.

4.1.19 System.ServiceProcess.ServiceControllerPermission

Allows to control accessing windows services. The class inherits from ResourcePermissionBase which in turn is inherited from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. The corresponding entry in configuration console is called “Service Controller”. Class is contained in module “System.ServiceProcess.dll”.

Parameters and Properties:

MachineName : String
Name of the machine where the service is running.

ServiceName : String
Name of the windows service itself.

PermissionsAccess : ServiceControllerPermissionAccess
Defines the type of access you want to allow. Can be set to None, Control and Browse:

- Browse - The ServiceController can connect to und view the status, but not control an existing service.
- Control - The ServiceController can connect and completely control the service.
- None – The ServiceController has no access to the service.

4.2 Permissions not represented in Configuration Console

4.2.1 System.Configuration.ConfigurationPermission

Provides a permission structure that allows methods or classes to access configuration files. The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 2.0. Class is contained in module “System.Configuration.dll”.

The new ConfigurationPermission CAS permission in 2.0 protects access to configuration files using the new Web/ConfigurationManager APIs. In ASP.NET only full and high trust levels have this permission.

Parameters and Properties: only default parameters (PermissionState)

4.2.2 System.Data.Common.DBDataPermission

Enables a .NET Framework data provider to help ensure that a user has a security level adequate for accessing data. The class is inherited directly from CodeAccessPermission. It is contained in .NET framework since version 1.0. Class is contained in module “System.Data.dll”. Builds the baseclass for the following Permission classes:

- System.Data.Odbc.OdbcPermission
- System.Data.OleDb.OleDbPermission
- System.Data.SqlClient.SqlClientPermission

Parameters and Properties:

AllowBlankPassword : Boolean
Gets a value indicating whether a blank password is allowed.

4.2.3 System.Data.Odbc.OdbcPermission

Enables the .NET Framework Data Provider for ODBC to help ensure that a user has a security level adequate to access an ODBC data source. The class is inherited from System.Data.Common.DBDataPermission which is itself derived from CodeAccessPermission and cannot be inherited. It is only supported in.NET framework since version 1.1, obsolete in all later versions. Class is contained in module “System.Data.dll”.

This class is intended for future use when the .NET Framework Data Provider for ODBC is enabled for partial trust scenarios. The .NET Framework Data Provider for ODBD currently requires FullTrust permission. At present, using the OdbcPermission class has no effect.

Parameters and Properties:

AllowBlankPassword : Boolean
Gets a value indicating whether a blank password is allowed.

Behavior : System.Data.KeyRestrictionBehavior
Defines access levels and can be set to AllowOnly and PreventUsage:

- AllowOnly - Default. Identifies the only additional connection string parameters that are allowed.
- PreventUsage - Identifies additional connection string parameters that are not allowed.

4.2.4 System.Data.OleDb.OleDbPermission

Enables the .NET Framework Data Provider for OLE DB to help ensure that a user has a security level adequate to access an OLE DB data source. The class is inherited from System.Data.Common.DBDataPermission which is itself derived from CodeAccessPermission and cannot be inherited. It is contained in.NET framework since version 1.0. Class is contained in module “System.Data.dll”.

This class is intended for future use when the .NET Framework Data Provider for OLE DB is enabled for partial trust scenarios. The .NET Framework Data Provider for OLE DB currently requires FullTrust permission. Currently, using the OleDbPermission class has no effect.

Parameters and Properties:

Provider : System.String
This property has been marked as obsolete. Setting this property will have no effect.

AllowBlankPassword : Boolean
Gets a value indicating whether a blank password is allowed.

4.2.5 System.Data.OracleClient.OraclePermission

Enables the .NET Framework Data Provider for Oracle to help ensure that a user has a security level adequate to access an Oracle database. The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.1. Class is contained in module "System.Data.OracleClient.dll".

The IsUnrestricted property takes precedence over the AllowBlankPassword property. Therefore, if you set AllowBlankPassword to false, you must also set IsUnrestricted to false to prevent a user from making a connection using a blank password.

Parameters and Properties:

AllowBlankPassword : Boolean
Gets a value indicating whether a blank password is allowed.

4.2.6 System.Net.Mail.SmtpPermission

Controls access to Simple Mail Transport Protocol (SMTP) servers. The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 2.0. Class is contained in module "System.dll".

Parameters and Properties:

Access : System.Net.Mail.SmtpAccess
Specifies the level of access allowed to a Simple Mail Transport Protocol (SMTP) server. Can be set to Connect, ConnectToUnrestrictedPort and None:

- Connect – Connection to an SMTP host on the default port (port 25).
- ConnectToUnrestrictedPort – Connection to an SMTP host on any port.
- None – No access to an SMTP host.

4.2.7 System.Net.NetworkInformation.NetworkInformationPermission

Controls access to network information and traffic statistics for the local computer. The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 2.0. Class is contained in module "System.dll".

Parameters and Properties:

Access : System.Net.NetworkInformation.NetworkInformationAccess
Specifies permission to access information about network interfaces and traffic statistics. This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.Can be set to None, Ping and Read:

- None – No access to network information.
- Ping – Ping access to network information.
- Read – Read access to network information.

4.2.8 System.Security.Permissions.DataProtectionPermission

This permission is used to control the ability to encrypt data and memory using the ProtectedData and ProtectedMemory classes. The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 2.0. Class is contained in module "System.Security.dll".

Parameters and Properties:

Flag : System.Security.Permissions.DataProtectionPermissionFlags
Specifies the access permissions for encrypting data and memory. This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values. Can be set to AllFlags, NoFlags, ProtectData, ProtectMemory, UnprotectData and UnprotectMemory:

- AllFlags – The ability to encrypt data, encrypt memory, unencrypt data, and unencrypt memory.
- NoFlags – No protection abilities.
- ProtectData – The ability to encrypt data.
- ProtectMemory – The ability to encrypt memory.
- UnprotectData – The ability to unencrypt data.
- UnprotectMemory – The ability to unencrypt memory.

Notes:

Many of these flags can have powerful effects and should be granted only to highly trusted code.

4.2.9 System.Security.Permissions.GacIdentityPermission

Defines the identity permission for files originating in the global assembly cache. The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 2.0. Class is contained in module "mscorlib.dll".

Files are either in the global assembly cache, or they are not. There are no variations to the permission granted, so all GacIdentityPermission objects are equal.

Parameters and Properties: only default parameters (PermissionState)

Notes:

In the .NET Framework versions 1.0 and 1.1, demands on the identity permissions are effective even when the calling assembly is fully trusted. That is, although the calling assembly has full trust, a demand for an identity permission fails if the assembly does not meet the demanded criteria. In the .NET Framework version 2.0, demands for identity permissions are ineffective if the calling assembly has full trust. This assures consistency for all permissions, eliminating the treatment of identity permissions as a special case.

4.2.10 System.Security.Permissions.KeyContainerPermission

Controls the ability to provide limited access to key containers. The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 2.0. Class is contained in module "mscorlib.dll".

Parameters and Properties:

Flags : System.Security.Permissions.KeyContainerPermissionFlags
Specifies the type of key container access allowed. This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values. Many of these flags can have powerful effects and should be granted only to highly trusted code. The most powerful of the flags are Create, Delete, Import, Export, Sign, Decrypt, and AllFlags. Can be set to the following values:

- AllFlags – Create, decrypt, delete, and open a key container; export and import a key; sign files using a key; and view and change the access control list for a key container.
- ChangeAcl – Change the access control list (ACL) for a key container.
- Create – Create a key container. Creating a key container also creates a file on disk. It is very important that any key container that is created is removed when it is no longer in use.
- Decrypt – Decrypt a key container. Decryption is a privileged operation because it uses the private key.
- Delete – Delete a key container. Deleting a key container can constitute a denial of service attack because it prevents the use of files encrypted or signed with the key. Therefore, deletion is a privileged operation.
- Export – Export a key from a key container. The ability to export a key is potentially harmful because it removes the exclusivity of the key.
- Import – Import a key into a key container. The ability to import a key can be as harmful as the ability to delete a container because importing a key into a named key container replaces the existing key.
- NoFlags – No access to a key container.
- Open – Open a key container and use the public key. Open does not give permission to sign or decrypt files using the private key, but it does allow a user to verify file signatures and to encrypt files. Only the owner of the key is able to decrypt these files using the private key.
- Sign – Sign a file using a key. The ability to sign a file is potentially harmful because it can allow a user to sign a file using another user's key.
- ViewAcl – View the access control list (ACL) for a key container.

AccessEntries : System.Security.Permissions.KeyContainerPermissionAccessEntry
Specifies access rights for specific key containers. This class cannot be inherited. Contains the following properties:

- Flags : System.Security.Permissions.KeyContainerPermissionFlags
  Gets or sets the key container permissions.
- KeyContainerName : String
  Gets or sets the key container name.
- KeySpec : Int32
  Gets or sets the key specification.
- KeyStore : String
  Gets or sets the name of the key store.
- ProviderName : String
  Gets or sets the provider name.
- ProviderType : Int32
  Gets or sets the provider type.

4.2.11 System.Security.Permissions.PublisherIdentityPermission

Represents the identity of a software publisher.The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. Class is contained in module "mscorlib.dll".

Parameters and Properties:

Certificate : System.Security.Cryptography.X509Certificates.X509Certificate
Gets or sets an Authenticode X.509v3 certificate that represents the identity of the software publisher.

Notes:

In the .NET Framework versions 1.0 and 1.1, identity permissions cannot have an Unrestricted permission state value. In the .NET Framework version 2.0, identity permissions can have any permission state value. This means that in version 2.0, identity permissions have the same behavior as permissions that implement the IUnrestrictedPermission interface. That is, a demand for an identity always succeeds, regardless of the identity of the assembly, if the assembly has been granted full trust.

In the .NET Framework versions 1.0 and 1.1, demands on the identity permissions are effective, even when the calling assembly is fully trusted. That is, although the calling assembly has full trust, a demand for an identity permission fails if the assembly does not meet the demanded criteria. In the .NET Framework version 2.0, demands for identity permissions are ineffective if the calling assembly has full trust. This assures consistency for all permissions, eliminating the treatment of identity permissions as a special case.

4.2.12 System.Security.Permissions.ResourcePermissionBase

Allows control of code access security permissions. The class is inherited directly from CodeAccessPermission. It is contained in .NET framework since version 1.0. Class is contained in module "System.dll". Builds the baseclass for the following Permission classes:

- System.Diagnostics.EventLogPermission
- System.Diagnostics.PerformanceCounterPermission
- System.DirectoryServices.DirectoryServicesPermission
- System.ServiceProcess.ServiceControllerPermission

When inheriting from ResourcePermissionBase, you must provide at least three constructors, set two properties, and provide a third property. The required constructors are: a default constructor, one that takes a PermissionState as a parameter, and as many as needed that take values for the properties. The properties that need to be set are PermissionAccessType and TagNames. The third property that is needed is one that returns the permission entries. For an example of an implementation of this class, see PerformanceCounterPermission. In PerformanceCounterPermission, the TagNames property is set privately to "Machine" and "Category", the PermissionAccessType property is privately set to the type of PerformanceCounterPermissionAccess, and the PermissionEntries property returns the permission entries.

Parameters and Properties:

Entry: System.Security.Permissions.ResourcePermissionBaseEntry
Defines the smallest unit of a code access security permission set.

PermissionAccessType : Type
Gets or sets an enumeration value that describes the types of access that you are giving the resource.

TagNames : String[]
Gets or sets an array of strings that identify the resource you are protecting. When you inherit from ResourcePermissionBase, you must set this property. For an example of an implementation of this class, see PerformanceCounterPermission. In PerformanceCounterPermission, the TagNames property is set privately to "Machine" and "Category".

Notes:

The ResourcePermissionBase class compares strings using ordinal sort rules and ignores the case of the strings being compared.

4.2.13 System.Security.Permissions.SiteIdentityPermission

Using this class, it is possible to ensure that callers are from a specific Web site The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. Class is contained in module "mscorlib.dll".

Parameters and Properties:

Site : System.String
Gets or sets the current site. Site identity is only defined for code from URLs with the protocols of HTTP, HTTPS, and FTP. A site is the string between the "//" after the protocol of a URL and the following "/", if present, for example, "www.fourthcoffee.com" in the URL "http://www.fourthcoffee.com/process/grind.htm".

This excludes port numbers. If a given URL is “http://www.fourthcoffee.com:8000/”, the site is “www.fourthcoffee.com”, not “www.fourthcoffee.com:8000”.

Sites can be matched exactly, or by a wildcard ("*") prefix at the dot delimiter. For example, the site name string “*.fourthcoffee.com” matches “fourthcoffee.com” as well as “www.fourthcoffee.com”. Without a wildcard, the site name must be a precise match. The site name string * will match any site, but will not match code that has no site evidence.

Notes:

In the .NET Framework versions 1.0 and 1.1, demands on the identity permissions are effective, even when the calling assembly is fully trusted. That is, although the calling assembly has full trust, a demand for an identity permission fails if the assembly does not meet the demanded criteria. In the .NET Framework version 2.0, demands for identity permissions are ineffective if the calling assembly has full trust. This assures consistency for all permissions, eliminating the treatment of identity permissions as a special case.

SiteIdentityPermission grants permission for all paths to the site, including both the URL and the IP address. To Deny access to a site, you must Deny all possible paths to the site. For example, if “www.fourthcoffee.com” is located at IP address 192.168.238.241, to Deny access to “www.fourthcoffee.com”, you must Deny “www.fourthcoffee.com”, 192.168.238.241 and any other path that you can use to access the site. A better technique to deal with multiple paths is to use a combination of PermitOnly and Deny. For more information on this subject and the use of PermitOnly with Deny, see Canonicalization Problems Using Deny.

In the .NET Framework versions 1.0 and 1.1, identity permissions cannot have an Unrestricted permission state value. In the .NET Framework version 2.0, identity permissions can have any permission state value. This means that in version 2.0, identity permissions have the same behavior as permissions that implement the IUnrestrictedPermission interface.

4.2.14 System.Security.Permissions.StrongNameIdentityPermission

Defines the identity permission for strong names. The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. Class is contained in module “mscorlib.dll”.

Use StrongNameIdentityPermission to achieve versioning and naming protection by confirming that the calling code is in a particular strong-named code assembly.
A strong name identity is based on a cryptographic public key called a blob optionally combined with the name and version of a specific assembly. The key defines a unique namespace and provides strong verification that the name is genuine, because the definition of the name must be in an assembly signed by the corresponding private key.
Note that the validity of the strong name key is not dependent on a trust relationship or any certificate necessarily being issued for the key.

Parameters and Properties:

PublicKey : System.Security.Permissions.StrongNamePublicKeyBlob
Gets or sets the public key blob that defines the strong name identity namespace.

Name : System.String
Gets or sets the simple name portion of the strong name identity.

Version : System.Version
Gets or sets the version number of the identity.

Notes:

Full demands for StrongNameIdentityPermission succeed only if all the assemblies in the stack have the correct evidence to satisfy the demand. Link demands using StrongNameIdentityPermissionAttribute succeed if only the immediate caller has the correct evidence.

4.2.15 System.Security.Permissions.UrlIdentityPermission

Defines the identity permission for the URL from which the code originates. The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. Class is contained in module “mscorlib.dll”.

The complete URL is considered, including the protocol (HTTP, HTTPS, FTP) and the file. For example, “http://www.fourthcoffee.com/process/grind.htm” is a complete URL.

URLs can be matched exactly or by a wildcard in the final position, for example: “http://www.fourthcoffee.com/process/*”. URLs can also contain a wildcard ("*") prefix at the dot delimiter. For example, the URL name string “http://www.fourthcoffee.com/process/grind.htm/” is a subset of “http://*.fourthcoffee.com/process/grind.htm/” and “http://*.com/process/grind.htm/”.

In the .NET Framework versions 1.0 and 1.1, demands on the identity permissions are effective even when the calling assembly is fully trusted. That is, although the calling assembly has full trust, a demand for an identity permission fails if the assembly does not meet the demanded criteria. In the .NET Framework version 2.0, demands for identity permissions are ineffective if the calling assembly has full trust. That is, a demand for an identity always succeeds, regardless of the identity of the assembly, if the assembly has been granted full trust. This assures consistency for all permissions, eliminating the treatment of identity permissions as a special case.

UrlIdentityPermission grants permission for all paths to the file, including both the URL and the IP address. To Deny access to the file, you must Deny all possible paths to the file. For example, if “http://www.fourthcoffee.com/process/grind.htm” is located at IP address 192.168.238.241, to Deny access to “http://www.fourthcoffee.com/process/grind.htm”, you must Deny “http://www.fourthcoffee.com/process/grind.htm”, 192.168.238.241/grind.htm and any other path that you can use to access the code. Unfortunately, there are a myriad of ways URL paths can be phrased, so it is extremely difficult to block all paths through the use of Deny alone. A better technique to deal with multiple paths is to use a combination of PermitOnly and Deny. PermitOnly allows you to identify a finite set of URLs that you can provide access to, and then Deny allows you to explicitly select addresses from that set that you want to deny access to.

In the .NET Framework versions 1.0 and 1.1, identity permissions cannot have an Unrestricted permission state value. In the .NET Framework version 2.0, identity permissions can have any permission state value. This means that in version 2.0, identity permissions have the same behavior as permissions that implement the IUnrestrictedPermission interface.

Parameters and Properties:

Url : String
Gets or sets a URL representing the identity of Internet code. The complete URL is considered, including the protocol (HTTP, HTTPS, FTP) and the file, for example: “http://www.fourthcoffee.com/process/grind.htm/”. URLs can be matched exactly or by a wildcard in the final position, for example: “http://www.fourthcoffee.com/process/*”.

4.2.16 System.Security.Permissions.ZoneIdentityPermission

Defines the identity permission for the zone from which the code originates. The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.0. Class is contained in module “mscorlib.dll”.
This permission can determine whether calling code is from a certain zone. Zones are configured according to the Microsoft Internet Explorer options, and are mapped from URL by Internet Explorer's IInternetSecurityManager and related APIs. Only exact zone matches are defined for the permission; a URL can only belong to one zone.

Parameters and Properties:

Zone : System.Security.SecurityZone
Gets or sets the zone represented by the current ZoneIdentityPermission. To configure zones, use the Security tab on the Microsoft Internet Explorer options panel. Can be set to Internet, Intranet, MyComputer, NoZone, Trusted and Untrusted:

- Internet – The Internet zone is used for the Web sites on the Internet that do not belong to another zone.
- Intranet – The local intranet zone is used for content located on a company's intranet. Because the servers and information would be within a company's firewall, a user or company could assign a higher trust level to the content on the intranet.
- MyComputer – The local computer zone is an implicit zone used for content that exists on the user's computer.
- NoZone – No zone is specified.
- Trusted – The trusted sites zone is used for content located on Web sites considered more reputable or trustworthy than other sites on the Internet. Users can use this zone to assign a higher trust level to these sites to minimize the number of authentication requests. The URLs of these trusted Web sites need to be mapped into this zone by the user.
- Untrusted – The restricted sites zone is used for Web sites with content that could cause, or could have caused, problems when downloaded. The URLs of these untrusted Web sites need to be mapped into this zone by the user.

Notes:

In the .NET Framework versions 1.0 and 1.1, demands on the identity permissions are effective, even when the calling assembly is fully trusted. That is, although the calling assembly has full trust, a demand for an identity permission fails if the assembly does not meet the demanded criteria. In the .NET Framework version 2.0, demands for identity permissions are ineffective if the calling assembly has full trust. This assures consistency for all permissions, eliminating the treatment of identity permissions as a special case.

4.2.17 System.Transactions.DistributedTransactionPermission

The permission that is demanded by System.Transactions when management of a transaction is escalated to MSDTC. The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 2.0. Class is contained in module “System.Transactions.dll”.

A local transaction only consumes resources from a single durable source or mutilple volatitle sources. A distributed transaction, on the other hand, has to interact with multiple durable resources, potentially across the network. When management of a transaction is escalated to MSDTC, the code that initiated the escalation is verified to have this permission.
The security demand affects the code that initiated the escalation, not necessarily the code that originally created the local transaction.

Parameters and Properties: only default parameters (PermissionState)

4.2.18 System.Web.AspNetHostingPermission

Controls access permissions in ASP.NET hosted environments. The class is inherited directly from CodeAccessPermission and cannot be inherited. It is contained in .NET framework since version 1.1. Class is contained in module “System.dll”.

The AspNetHostingPermission class is used in conjunction with code access security to help protect public types in the System.Web namespaces. Code must be assigned at least the Minimal trust level to access protected ASP.NET classes.

The AspNetHostingPermission class's Level property is set by configuring the appropriate trust level in the trust configuration element. By default, the level attribute of the trust configuration element is set to Full. That is, by default, ASP.NET applications run under the Unrestricted level. When an ASP.NET application domain

is created, ASP.NET reads the value specified for the level attribute of the trust configuration element, creates an instance of the AspNetHostingPermission class with the specified Level attribute, and then adds the class to the permission set for the application domain. For more information, see ASP.NET Trust Levels and Policy Files.

It is recommended that you set the level attribute of the trust configuration element to High for sites that are trusted. For sites that are not trusted, such as a Web server that hosts sites that run code from an external customer, it is recommended that you set the level attribute of the trust configuration element to Medium.

The permission sets that are defined by default for the .NET Framework (for example, LocalIntranet, Internet, and so on) do not include the AspNetHostingPermission permission. That is, the AspNetHostingPermission permission is only assigned, by default, to applications that are running under Full trust.

Parameter and Properties:

Level: System.Web.AspNetHostingPermissionLevel
Specifies the trust level that is granted to an ASP.NET Web application. The members of this enumeration define application security levels ranging from full trust (the application is unconstrained by code access security) to minimal trust (the application has permission only to execute). You set the trust level for an ASP.NET resource with the trust configuration element in a Web.config or Machine.config file. Can be set to High, Low, Medium, Minimal, None and Unrestricted:

- High – Indicates that features protected with a demand for any level less than or equal to the High trust level will succeed. This level is intended for highly trusted managed-code applications that need to use most of the managed permissions that support semi-trusted access. It does not grant some of the highest permissions (for example, the ability to call into native code), but it does provide a way to run trusted applications with least privilege or to provide some level of constraints for highly trusted applications. This level is granted by configuring at least the High trust level in the trust section in a configuration file.
- Low – Indicates that features protected with a demand for any level less than or equal to the Low level will succeed. This level is intended to allow read-only access to limited resources in a constrained environment. This level is granted by specifying the Low trust level in the trust section in a configuration file.
- Medium – Indicates that features protected with a demand for any level less than or equal to the Medium level will succeed. This level is granted by configuring at least the Medium trust level in the trust section in a configuration file.
- Minimal – Indicates that features protected with a demand for the Minimal level will succeed. This level allows code to execute but not to interact with resources present on the system. This level is granted by configuring at least the Minimal trust level using the trust section in a configuration file.
- None – Indicates that no permission is granted. All demands for AspNetHostingPermission will fail.
- Unrestricted – Indicates that all demands for permission to use all features of an application will be granted. This is equivalent to granting Full trust level in the trust section in a configuration file.

Notes:

If you want to lock security policy for a specific server or Web site, use the location element (of configuration file) in conjunction with the allowOverride=false attribute to ensure that trust policy settings cannot be overridden locally.

- [0] CLR-Sicherheitsrichtlinien auf Clients installieren (by Michael Willers)
  http://www.dotnetpro.de/articles/onlinearticle1697.aspx

- [1] www.leastprivilege.com (by Dominick Baier)
  http://www.leastprivilege.com

- [2] Code Access Security in the .NET Framework (by Enrico Sabbadin)
  http://www.devx.com/vb2themax/Article/19886

- [3] Bill Bozeman's Blog on .NET and C#:
  http://www.bozemanblog.com

- [4] Understanding .NET Code Access Security (by UB)
  http://www.codeproject.com/dotnet/UB_CAS_NET.asp

- [5] MSDN Library
  http://msdn.microsoft.com/library/

- [6] .Net Security Blog - Console Applications require UIPermission (by Shawn Farkas)
  http://blogs.msdn.com/shawnfa/archive/2005/06/06/425804.aspx

- [7] Security Briefs (by Keith Brown)
  http://pluralsight.com/blogs/keith/

- [8] Was darf mein Code? - Das Sicherheitsmodell der Common Language Runtime (by Michael Willers)
  http://www.microsoft.com/germany/msdn/library/security/WasDarfMeinCodeDasSicherheitsmodellDerCommonLanguageRuntime.mspx?mfr=true

- [9] Managing .NET Code Access Security (by Joe Mayo)
  http://www.eps-publishing.com/article.aspx?quickid=0405031&page=1

- [10] Code Access Security (by Silan Liu)
  http://progtutorials.tripod.com/Code_Access_Security.htm

- [11] Code Access Security in Practice (by J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla and Anandha Murukan)
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh08.asp

- [12] Using Code Access Security with ASP.NET
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh09.asp

- [13] Security Enhancements in the .NET Framework 2.0 (by Keith Brown)
  http://msdn.microsoft.com/msdnmag/issues/05/01/SecurityBriefs

- [14] ConfigurationPermission and requirePermission (by Dominick Baier)
  www.leastprivilege.com/ConfigurationPermissionAndRequirePermission.aspx

- [15] .Net Security Blog - Reading a File from Partial Trust (by Shawn Farkas)
  http://blogs.msdn.com/shawnfa/archive/2005/03/30/403896.aspx

- [16] Visual Basic .NET Code Security Handbook (by Eric Lippert)